VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2019 Lukáš Středula

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Lukáš Středula**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: ArcelorMittal Ostrava, a.s.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Mgr. Pavla Dráždilová, Ph.D.**

Konzultant bakalářské práce: Mgr. Martin Bagier

Datum zadání: 01.09.2018

Datum odevzdání: 30.04.2019

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 23. dubna 2019

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 23. dubna 2019

ArcelorMittal Ostrava a.s.
Vratimovská 689, 707 02 Ostrava 7
45 - Automatizace
MES a procesní modelování

Abstrakt

Bakalářská práce popisuje průběh individuální odborné praxe ve firmě ArcelorMittal Ostrava a.s. na pozici programátora. Praxe byla zaměřena na návrh, tvorbu řešení pro sběr, vyhodnocení informací a logů z různých systémů. Úvodem jsou uvedeny informace o firmě. Následují technologie, které byly použity při vyvíjení aplikace. Požadovaná aplikace byla zaměřená na dvě oblasti, sledování výkonu a logování. Sledování výkonu se zaměřuje na operační systém, procesy, databáze a jejich následné optimalizace. Oblast logování obsahuje zejména získávání Windows událostních logů a zavedení jednotného standardu. V rámci sledování výkonu byly primárně použity výkonnostní metriky. Tyto metriky představovaly výkonnostní čítače, které jsou součástí operačního systému Windows. Díky těmto metrikám jsme schopni ohodnotit výkonnost na základě několika parametrů jaké jsou např. rychlost, propustnost, latence, využití periférií. Většina získaných dat byla ukládána do NoSQL databáze a následně vizualizována. Získaná data nám zajistila potřebné informace pro lepší a kvalitnější správu v rámci automatizace procesů firmy. Na závěr si uvedeme přínos bakalářské práce pro mne, získané zkušenosti ve firmě a také můj přínos pro firmu díky realizaci této bakalářské praxe.

Klíčová slova: monitorování, logování, optimalizace, výkonnost

Abstract

The aim of this bachelor thesis is to describe the course of individual bachelor practise at ArcelorMittal Ostrava a.s. company at the position of programmer. The practise was focused on design and development of a solution for collecting and evaluation of information and log files from various systems. First there are stated some basic facts about the company, followed by the description of technologies used for the final solution. The solution is composed of two main areas – performance monitoring and logging. Performance monitoring is focused on operation system, processes, databases and follow-up optimization. The logging part of the solution contains mainly obtaining information from Windows Events Viewer and consecutive implementation of one logging standard. Performance monitoring was evaluated by performance metrics which are calculated from performance counters (these are part of operation system). The performance metrics help us to evaluate the performance with use of several indicators such as frequency of CPU, throughput, latency or periphery usage. Most of the obtained data was stored into NoSQL database and then visualised. These data provided information necessary for optimized management of process automation. In conclusion of the thesis I state main benefits of the practise both for myself and for the company.

Key Words: monitoring, logging, optimization, performance

Obsah

Seznam použitých zkratek a symbolů

SQL	–	Structured Query Language
JSON	–	JavaScript Object Notation
API	–	Application Programming Interface
GUI	–	Graphical User Interface
IP	–	Internet Protocol
WMI	–	Windows Management Instrumentation
CIM	–	Common Information Model
OOP	–	Objektově orientované programování
COM	–	Component Object Model
IIS	–	Internet Information Services
CPU	–	Centrální procesorová jednotka
RDBMS	–	Relational database management system
IoT	–	Internet of Things
PC	–	Personal computer
OS	–	Operační systém
ID	–	IDentification

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Bakalářská práce je založena na absolvování odborné individuální praxe ve firmě ArcelorMittal Ostrava a.s. (dále pouze ArcelorMittal Ostrava). Podmětem pro vznik bakalářské práce byla možnost využití informačních a komunikačních technologií pro implementaci výsledné aplikace. Velmi často se dostáváme do situací, kdy nemáme vůbec kontrolu a přehled nad spuštěným systémem. Nejsme tak schopni v případě vzniklých problémů rychle a efektivně jednat. Aplikace pro sbírání logů bývají často různorodé, proto je potřeba zavést jednotný systém. Ve firmě ArcelorMittal Ostrava je mnoho automatizovaných procesů a velké množství informací, které je nutné sledovat. Vzhledem k velkému množství dat je zapotřebí mnoho serverů a měřících zařízení pro kvalitní výrobu firmy. Z těchto důvodů potřebujeme zavést logování a monitorování produkčních aplikací a systémů.

Nabízí se získávat informace, které nám pomohou zlepšit přehlednost a funkčnost operačního systému, databází, aplikací a procesů. Je požadováno monitorovat systémy a centralizovat sběr logů pro dané prostředí. V oblasti logování požadujeme získávání Windows událostních logů, které generuje operační systém, a informace o produkčním systému. V oblasti monitorování se zaměříme na výkonnostní metriky. Operační systém Windows nabízí výkonnostní čítače, které jsou již součástí operačního systému Windows od verze Windows 2000. Nasbírané informace jsou ukládány do NoSQL databáze Elasticsearch. Data jsou následně vizualizována prostřednictvím nástroje Kibana. Díky těmto zpracovaným informacím jsme schopni identifikovat a optimalizovat vzniklé problémy.

Obsah dané problematiky je rozsáhlý. Bylo nutné nastudovat a shromaždit velké množství informací, které vedly k zpracování a vyřešení bakalářské práce. Tyto faktory se odrazily i na potřebném času, který byl nutný věnovat tématu. V době vzniku tohoto tématu jsem již pracoval ve firmě ArcelorMittal Ostrava. Jelikož jsem měl o téma zájem, bylo mi umožněno téma zpracovat v rámci bakalářské praxe. Téma je z mého pohledu velice zajímavé a nabízí nejenom možnost jak se zlepšit v programování, ale také v oblasti analýzy a optimalizace systémů.

2 Představení firmy ArcelorMittal Ostrava a.s.

ArcelorMittal Ostrava je hutnická firma patřící do největší světové ocelářské a těžařské skupiny ArcelorMittal. Společnost působí v areálu Nové hutě v jižní části Ostravy. Roční kapacita výroby je 3 miliony tun oceli. Kromě tuzemského trhu prodává společnost své výrobky do více než 40 zemí světa. ArcelorMittal Ostrava a její dceřiné společnosti mají 7 250 zaměstnanců [5]. Výrobní činnost je zaměřena především na výrobu a zpracování surového železa, oceli a hutní druhovýrobu. Největší podíl hutní výroby tvoří dlouhé a ploché válcované výrobky.

Součástí firmy je také IT oddělení, které využívá termín „in-house software development“ u vývoje a programování vlastních aplikací. Požadovaný software je vyvíjen interně v rámci společnosti. Tento způsob vyvíjení je založen na konceptu využívání zaměstnanců firmy (zpravidla se jedná o tým analytiků a programátorů), kteří jsou schopni vyvíjet a implementovat informační systémy a aplikace. Výsledné systémy jsou navrženy na míru firmy. Tata organizace vývoje přináší spoustu výhod, jako jsou např. plná kontrola nad funkčností, možnost implementovat specifické požadavky firmy a rozšiřitelnost.

2.1 Pracovní zařazení

Ve firmě ArcelorMittal Ostrava a.s. pracuji již od dubna 2016, kdy jsem nastupoval na pozici programátora. Konkrétně do závodu automatizace, který spolupracuje se závodem koksovny. Standardní práce IT oddělení závodu automatizace je většinou pokryta pomocí systému Z10MES. Jedná se o interní systém, který je vyvíjen uvnitř společnosti, kde veškerá práva a licence patří právě této firmě. V rámci našeho týmu se staráme o samotný vývoj a podporu spojenou s tímto systémem. Tento systém slouží pro sledování provozního chodu, jeho vyhodnocování a řízení výroby koksovny. Já aktuálně pracuji v týmu 5 lidí, který se skládá z programátorů, kde každý člen má specifickou roli a schopnosti.

2.2 Orientační čas strávený u daných oblastí

- Nastudování vhodné technologie pro logování - 16 hodin.
- Nastudování technologie RabbitMQ - 24 hodin.
- Nastudování vhodné technologie pro ukládání dat - 24 hodin.
- Nastudování technologie Kibana - 16 hodin.
- Nastudování technologie PowerShell - 10 hodin.
- Problematika logování - 32 hodin.
- Problematika sledování výkonu (procesů a operačního systému) - 48 hodin.
- Problematika databází - 32 hodin.

– Implementace a zavedení standardu pro událostní logy - 60 hodin.

– Implementace a zavedení standardu pro sledování výkonu - 120 hodin.

– Optimalizase aplikace - 50 hodin.

3 Použité technologie

Během bakalářské praxe byly použité technologie, které nám zajistily potřebnou funkcionalitu pro implementování požadavků pro sběr logů a monitorování produkčních systémů. Většina implementačních věcí byla prováděna ve vývojovém prostředí Microsoft Visual Studio 2017 ve verzi 15.8.9. V tomto prostředí byl použit OOP jazyk C#, který byl spojený s vybranými .NET frameworky.

Pro zaznamenávaní informací o běhu systému byl použit logovací nástroj log4net. Na tento nástroj se váže použití RabbitMQ. Tato technologie byla využita jako prostředník mezi log4Net a Elasticsearch pro větší flexibilitu a rozšiřitelnost. Pro ukládání získaných informací o systémech a serverech jsme využili NoSQL databázi Elasticsearch. Pro následnou vizualizaci dat byl použit vizualizační nástroj Kibana. Obě tyto technologie jsou od firmy Elastic. V neposlední řadě jsme použili také rozšiřitelný textový nástroj PowerShell a relační databázový systém MS SQL Server.

3.1 C#

Je vysokoúrovňový objektově orientovaný programovací jazyk vyvinutý firmou Microsoft. Tento jazyk je založen na jazycích C++ a Java [1].

Základní vlastnosti jazyka jsou:

- neexistující vícenásobná dědičnost, každá třída může být potomkem pouze jedné třídy,
- neexistují žádné globální proměnné a metody,
- „property“, které zapouzdřují do své definice „Getter“ a „Setter“,
- nepotřebuje dopředné deklarace,
- jazyk je „case sensitive“, tzn. rozlišuje velká a malá písmena.

C# je navržen pro psaní aplikací jak pro zařízení se sofistikovanými operačními systémy, tak pro zařízení s omezenými možnostmi, např. IoT zařízení. Proto tento jazyk lze např. využít k tvorbě databázových programů, webových aplikací, webových služeb, formulářových aplikací ve Windows, softwaru pro mobilní zařízení. Technologie C# byla použita ve verzi 7.0

3.2 log4net

Jedná se o logovací nástroj, který pomáhá zaznamenávat informace do předem zvoleného typu úložiště. Logovací nástroj se řadí do skupiny logovacích frameworků a byl vyvinut neziskovou organizací Apache Software Foundation [2].

Hlavními přednostmi tohoto frameworku jsou rychlost a flexibilita. Framework nabízí několik možností nastavení. Jako hlavní si můžeme uvést výstupní úložiště, minimální úroveň, kterou chceme zaznamenávat, a výstupní formát logu. Je možné logovat do databáze, souboru nebo jen

zaznamenávat výpisy logů pro dané běhové prostředí. Tento framework je možné rozšířit o implementaci vlastního „Appenderu“, který umožní ukládaní a formátováni dle námi zvoleného vzoru.

Framework obsahuje metody, které slouží k zaznamenání informace s požadovanou úrovní uložení logu. Metody nabízí možnost zalogování objektu v datovém typu „string“ nebo „object“. Díky této vlastnosti je možné uložit objekt jakéhokoliv datového typu. Námi specifikovaný výsledný log pak má vlastosti, které přímo vyhovují našim potřebám. Technologie log4net byla použita ve verzi 2.0.8.

3.3 RabbitMQ

RabbitMQ je otevřený software implementující fronty zpráv [3]. Jde o nástroj, který byl napsán v jazyce Erlang pod záštitou společnosti Pivotal Software. RabbitMQ je multiplatformní, je proto možné ho využít ve všech známých programovacích jazycích jako jsou C#, Java, Python, PHP.

RabbitMQ je široce konfigurovatelný a flexibilní, což umožňuje programátorům zvolit si takové vlastnosti, které nejlépe vyhovují potřebám implementované služby. Základní motivací pro použití tohoto nástroje je mít prostředníka mezi producentem a příjemcem. Právě tohoto prostředníka předtavuje RabbitMQ. Můžeme si představit některé případy jako jsou: poslání emailu, komunikace s platební bránou, komunikace s externím pokladním API. Všechny tyto operace zabírají nějaký čas v systému. Proto se doporučuje takové události zpracovávat asynchronně, na pozadí, tak abychom nepřetěžovali systém. V našem případě jsme často využívali nějakou zprávu (v aplikaci zpravidla objekt), která je vygenerována v aplikaci a následně poslána do fronty. Princip je následující: producent pošle zprávu do fronty, kde je uchovávána tak dlouho, dokud se o ni nepřihlásí příjemce. Toto byla jedna ze základních strategií, která technologie RabbitMQ nabízí.

Kromě základní verze existuje ještě řada dalších konceptů pro tuto technologii. Jelikož fronta nemusí být pouze jedna a také příjemců může být více. Je možné, aby jedna fronta byla odebírána více příjemci, ale také každá fronta může mít svého vlastního příjemce. Z těchto vlastností můžeme pozorovat, že interní struktura RabbitMQ je poněkud složitější. Mezi frontou a producentem se ještě nachází výměnce. Ten přijímá zprávy od producenta a posílá je do front na základě toho, o jaký typ výměny se jedná.

První typ je „direct“. Ten doručuje zprávy do fronty na základě směrovacího (routovacího) klíče, který slouží pro výběr správné fronty. Druhý typ je „topic“. I zde se používá klíč uložený ve zprávě. Tento klíč se porovnává s regulárními výrazy specifikovanými v konfiguraci směrovače. Ve chvíli, kdy klíč odpovídá nějakému regulárnímu výrazu, je zpráva přesměrována do příslušné fronty. Třetí typ se jmenuje „headers“, který používá tzv. hlavičky, ty jsou připojené ke zprávě. To umožňuje detailnější konfiguraci směrování. Poslední je „fanout“. Tento typ směruje zprávy do všech front. Nemáme tak kontrolu nad posláním zprávy do potřebné fronty. Technologie RabbitMQ byla použita ve verzi 3.7.5.

3.4 Elasticsearch

Elasticsearch je vyhledávací a analytický otevřený software, kde jádro je založeno na knihovně Lucene. Byl napsán v jazyce Java, společností Elastic. Řadí se do skupiny NoSQL databází. Elasticsearch představuje hlavní produkt ze skupiny Elastic Stack, který slouží pro ukládání dat. Poskytuje vyhledávání na základě indexů. Tato vlastnost zajistí, že nalézení libovolného slova je provedeno v nejkratším možném čase. Přístup je skrze webové rozhraní HTTP. Uložená data jsou reprezentována v JSON formátu. Software je multiplatformní a mezi oficiální klienty patří C#, Java, PHP, Python a mnohé další jazyky.

Elasticsearch je nezávislý na struktuře ukládaných dat, je tedy možné ukládat jakýkoliv objekt [4]. Tato vlastnost je pro nás klíčová v ohledu na potřebu ukládat různé datové typy objektů do úložiště. Při vyvíjení produktu bylo cíleno na několik klíčových parametrů, jako jsou:

- rychlost,
- škálovatelnost,
- flexibilita,
- různorodé možnosti při hledání dat.

Elasticsearch je koncipován, tak aby si uměl hledat klíčová slova a to i při velké míře procházených dat. Uvedeme si vybrané případy, se kterými si Elasticsearch umí dobře poradit s ohledem na rychlost a výkonnost:

- hledání konkrétního výrazu slova při velkém míře dat,
- automatické doplňování hledaného slova na základě předchozích hledání.

Technologie Elasticsearch byla použita ve verzi 6.6.0.

3.5 Kibana

Kibana je otevřený software, který slouží k vizualizaci dat. Software byl napsán v jazyce JavaScript a řadí se také k produktům od společnosti Elastic. Kibana požadavkům spojených s vizualizací dat z NoSQL databáze Elasticsearch. Hlavní využití této technologie je pro hledání, zobrazování a interakci s daty, které jsou uložené v Elasticsearch indexech. Všechny tyto interakce jsou přes webové rozhraní.

Kibana nám umožňuje použití vizualizačních komponent, pomocí kterých lze vytvářet vizualizační okna (dashboardy). Komponenty použité v jednotlivých vizualizačních oknech mohou být různé variace grafů (sloupcový, koláčový, spojnicový), tabulek, map, časových průběhů. Při tvorbě těchto oken nabízí Kibana agregování uložených dat. Tato funkce je velmi užitečná a jsme schopni data agregovat podle vlastních požadavků. Možnosti agregace jsou např. průměr,

počet, maximum, minimum, poměr filtru, proceto. Můžeme pak dostávat data: za jednotlivé časové období, zjistit maxima a minima hodnot, celkové počty, průměrné hodnoty a celkové počty. Technologie Kibana byla použita ve verzi 6.6.0.

3.6 MS SQL Server

Jedná se relační databázový systém, který byl vyvinut společností Microsoft. Databázový systém byl naprogramován v jazyce C a C++. Tento systém primárně slouží pro ukládání dat. Systém nabízí pro přístup a práci s daty dvě možnosti použití. První je lokálně a druhá pak vzdáleně. Pro připojení k databázi vzdáleně potřebujeme IP adresu zařízení a přihlašovací údaje k databázi.

Pro práci s daty (dotazování, ukládání) slouží SQL jazyk. Kromě SQL dotazů můžeme využít nadstavby za pomocí T-SQL, která nám umožní např. implementovat procedury, které se ukládají na straně serveru. SQL dotazy můžeme psát skrze GUI, které nám zajistí např. SQL Server Management Studio. Druhou možností je implementace dotazů v aplikaci, kde je potřeba pracovat s daty. Musíme však opět znát přihlašovací údaje pro připojení do databáze a mít znalost SQL jazyka, případně využít framework, který dotazy vygeneruje za nás. MS SQL Server je nabízén ve 4 edicích:

- enterprise,
- standard,
- express,
- developer.

Technologie MS SQL Server byla použita ve verzích 11.0.6607.3 a 13.0.2216.0.

3.7 Windows PowerShell

Windows PowerShell je rozšiřitelný textový interpret příkazů se skriptovacím jazykem. Tento interpret byl vyvinut společností Microsoft. PowerShell je postavený na rozhraní .NET Frameworku. Díky tomuto rozhraní obsahuje Windows PowerShell objektovou rouru a rovněž velké množství funkcí pro správu pomocí „cmdlets“. Jedná se o specializované třídy .NET implementující určitou operaci. Windows PowerShell je nástupcem příkazového řádku Windows, tudíž dokáže pracovat s klasickými aplikacemi Windows (net.exe, ping.exe), ale také dokáže vytvářet instance libovolné .NET třídy, případně COM objekt.

Windows PowerShell na rozdíl od příkazového řádku dokáže přistupovat nejenom k souborovému systému, ale také např. k registrům systému, úložišti certifikátů. Toto umožňuje systém poskytovatelů, které tuto funkcionalitu přidávají. PowerShell také obsahuje podporu hostování v libovolné aplikaci, takže je možné implementovat podporu do různých aplikací, např. do Microsoft SQL Server 2008, IIS 7.Technologie Windows PowerShell byla použita ve verzi 5.1.17134.590.

4 Popis problematiky logování a měření metrik

Logování je druh techniky, který umožňuje získávat informace o systému. Logy zpravidla obsahují záznam činnosti, kterou systém vykonával. Tyto informace nám pak pomáhají k lepší správě a kontrole systému. Rovněž slouží k získání informací při diagnostice a řešení vzniklých chyb v rámci daného systému.

Pro lepší vysvětlení je uvedený příklad systému, který slouží jako eshop. Tento eshop může být spuštěn již v produkčním prostředí a nebylo by možné verzi ladit. V prvním kroku si zákazník naplní košík, vyplní všechny potřebné formuláře a následně bude chtít odeslat objednávku. Po té zmáčkne tlačítko odeslat a zobrazí se mu vizualizační prvek s nápisem „Objednávka se odesílá“. Pokud se stane, že proces trvá delší dobu např. 30 sekund, tak se zákazníkovi zobrazí nápis „Objednávka byla neúspěšně objednaná“. Zákazník je tedy nespokojen a v lepším případě si stěžuje na eshop. V horším případě tuto chybu nikdo ani nezaregistruje a přijdeme nejen o zákazníka, ale také o případný zisk z objednávky. Tento příklad demonstruje důležitost logování procesů v systému. Pokud bychom měli zavedené logování systému, tak řešení problémů bude otázkou několika minut. V opačném případě bychom vůbec nevěděli, co se v systému stalo špatně a kde máme hledat chybu.

Pro kvalitní a efektivní získávání informací o systému je vhodné si určit standard, který nám bude určovat základní linii logování. Je vhodné si určit, jaké informace chceme získávat, kde se budou ukládat a v jakém formátu je budeme chtít reprezentovat. Klíčový aspekt spočívá ve vybrání informací, které budeme chtít logovat. Není vhodné sbírat každý krok daného systému, ale také není ideální opak. Není doporučeno sbírat informace typu:

- spuštění systému,
- ukončení aplikace,
- provedení jednoduché činnosti (např. inkrementace hodnot).

Je proto vhodné udělat analýzu důležitosti a nedůležitosti informací.

V dnešní době informačních technologií se klade velký důraz na spolehlivost, výkonnost a efektivitu. Proto je vhodné zvolit metriky, které nám tyto informace poskytují. Pro ohodnocení míry výkonu a námi vybraných vlastností byly použity výkonnostní čítače. Existuje mnoho čítačů, které se zaměřují na různé kategorie a oblasti monitorování. V bakalářské práci se nacházejí námi vybrané čítače pro sledování databází, procesů a hardwarových komponent. Výkonnostní čítače umoňovaly monitorovat jednotlivé části jako jsou:

- procesory,
- operační paměti,
- pevné disky,

– síťová rozhraní,

– procesy,

– vybrané parametry databází.

Informace, které byly následně získány, tak pomohly nejen k optimalizaci samotného výkonu, ale také k lepší detekci úskalí a omezení v rámci celého systému.

Výkonnostní čítače jsou dostupné na operačních systémech Windows od verze Windows 2000 a novějších. Pokud bychom se chtěli podívat na tyto čítače, tak nám stačí si otevřít aplikaci „Sledování výkonu“. V této aplikaci se po spuštění ukáže základní přehled o fyzickém disku, procesoru, operační paměti a rozhraní sítě. Pokud si budeme chtít zobrazit detailnější informace o konkrétních čítačích, tak otevřeme záložku „Sledování výkonu“ v kategorii „Nástroje pro sledování“ a následně při kliknutí na ikonu „přidat“ se otevře nové okno obsahující všechny dostupné výkonnostní čítače v daném operačním systému.

4.1 Událostní logy

Událostní logy zaznamenávají jednotlivé události odehrávající se v systému. Tyto záznamy uchovávají informace obsahující jednotlivé činnosti tak, aby bylo možné pochopit aktuální chování systému a diagnostikovat problémy. Jsou důležité pro pochopení toho, co se děje v komplexních systémech. Především v takových, kde je velmi malá interakce mezi uživatelem a systémem (např. serverové aplikace, služby).

Je užitečné zkombinovat jednotlivé logy z několika různých zdrojů. Tento přístup v kombinaci se statistickou analýzou může nalézt vzájemné vztahy mezi rozdílnými událostmi na různých serverech.

4.1.1 Prohlížeč událostí (eventvwr.exe)

Jedná se aplikaci, která je součástí operačního systému Windows. Tato aplikace umožňuje zobrazit protokoly událostí, které se vyskytly za běhu operačního systému. Událostní logy, které spadají do protokolu systému Windows se řadí do 5 kategorií. Tyto kategorie se skládají z aplikací, zabezpečení, instalací, systému a předaných událostí. Operační systém třídí logy spadající do kategorie systém a aplikace do 4 úrovní:

– Kritická
  Úroveň označuje, že došlo k závažné poruše, která byla způsobena aplikací nebo komponentou operačního systému.

– Chyba
  Tato úroveň označuje, že došlo k problému, který může mít vliv na funkčnost aplikace (komponenty). Tento problém může být vyvolán také jinou aplikací (komponentou) než tou, ve které se nefunkčnost projevila.

– Upozornění
  Tato úroveň označuje, že došlo k problému, který může mít dopad na službu. Upozornění oznamuje, že může dojít k závažnějším problémům, pokud není provedno opatření.

– Informace
  Tato úroveň označuje, že došlo ke změně aplikace nebo komponenty. Například úspěšně se provedla operace a zdroj byl vytvořen nebo služba byla spuštěna.

– Podrobnosti („Verbose“)
  Úroveň nejčastěji označuje informaci o statusu provedené činnosti. Např. vývoj činnosti nebo úspěšné přijatí zprávy.

4.2 Metriky operačního systému

Výkonnostní čítače sloužily jako metriky pro monitorování námi vybraných parametrů. Představíme si tyto čítače, abychom si udělali základní představu, kdy a k čemu jsou vhodné. Čítače byly rozděleny do kategorií v závislosti na hardwarových komponentách. Jednotlivé čítače byly uvedeny v angličtině a to ze dvou důvodů:

1. OS jakékoliv jazykové verze dokáže implementovat čítače s anglickou verzí.
2. Většina serverů měla nainstalována OS Windows v anglické verzi jazyka.

Jednotlivé kategorie čítačů jsou rozdělené na základě hardwarových komponent, proto je použité následující schéma popisu:
„název čítače“ [„jednotky“]: \„název kategorie čítače“\„konkrétní název použitého čítače“ a následuje slovní popis činnosti čítače. V případech, kde je nutné specifikovat jméno instance, o které se mají data získávat, tak je napsáno v závorce (za názvem kategorie čítače). Toto schéma je aplikované na čítače v této kapitole a v kapitole 4.3.

1. **Procesor**

   - Procentuální využití procesoru [%]: \Processor(_Total)\% Processor Time
     Čítač, který ukazuje celkové vytížení procesoru v procentech. Výsledné procento se získává zprůměrováním všech instancí procesorů na základě průměrného času zaneprázdění procesorů.

     Kromě celkového ukazatele vytížení procesoru se také využívá sledování zatížení jednotlivých instancí procesorů [6]. Pro získávání těchto informací stačí vyměnit za parametr _Total konkrétní instaci procesoru. Např. \Processor(0)\% Processor Time.

   1.1. **Systémové statistiky**
     Procesor běží pod operačním systémem Windows ve dvou režimech a to v uživatelském a kernel (privilegovaném) režimu. Procesor se přepíná mezi těmito dvěma

režimy na základě typu kódu běžícího na procesoru. Aplikace běží v uživatelském módu a naopak komponenty OS běží v kernel módu.

- Procento využití procesoru v privilegovaném režimu [%]: \Processor(_Total) \% Privileged Time
  Tento čítač nám ukazuje procentuální čas, které procesor stráví spouštěním vláken v kernel (privilegovaném) režimu. Například pokud je volána služba OS Windows, která potřebuje přistoupit k datům, které jsou vyhrazené OS, tak je často volána v tomto režimu [6].
- Procento využití procesoru v uživatelském režimu [%]: \Processor(_Total)\% User Time
  Udává nám čas procesoru strávený v uživatelském režimu. Je to režim především vyhrazený pro aplikace, integrální podsystémy a podsystémy prostředí. Například pokud spustíme uživatelskou aplikaci, tak Windows vytvoří proces pro tuto aplikaci. Tento proces je pak navázán s vyhrazeným prostorem pro virtuální adresy a tabulku pro obsluhu.
- Délka fronty procesoru: \System\Processor Queue Length
  Tento čítač nám ukazuje počet vláken, které jsou ve frontě procesoru a čekají na spuštění. U vícejádrových procesorů platí, že je také pouze jedna fronta [9].

2. **Fyzická paměť (RAM)**

- Množství volné paměti [MB]: \Memory\Available MBytes
  Tento čítač zobrazuje velikost fyzické paměti v MB, které jsou okamžitě k dispozici pro přidělení procesu nebo pro interní použití systémem. Tato velikost se skládá ze součtu velikosti paměti mezipaměti, volných a vynulovaných stránek [6].
- Množství obsazené paměti všemi procesy: \Process(_Total)\Working Set
  Čítač, který nám zobrazuje množství použité paměti všemi procesy v bajtech. S ohledem na velikost operačních paměti je doporučeno převádět výslednou hodnotu na MB.
- Množství bajtů v mezipaměti [B]: \Memory\Cache Bytes
  Udává nám množství bajtů v mezipaměti (v systémové části) počítače. Skládá se z velikosti aktivních a uložených systémových souborů.
- Velikost nestránkovaného fondu [B]: \Memory\Pool Nonpaged Bytes
  Ukazuje velikost oblasti virtuální paměti systému. Tato oblast je určená pro objekty, které nelze zapsat na disk, ale musí po celou dobu vyhrazení zůstat ve fyzické paměti.

2.1. **Stránkování**

Jedná se o tzv. schéma správy paměti, podle které počítač ukládá a získává data ze sekundárního úložiště (pevného disku) pro následné použití v primární paměti (fyzické paměti).

- Stránky za sekundu: \Memory\Pages/sec
  Čítač obsahuje počet stránek čtených z disku nebo zapisovaných na disk, které mají vyřešit hardwarové chyby stránkování. Jedná se jeden z hlavních ukazatelů chyb způsobujících opožďování celého systému.
- Vstupní stránky za sekundu: \Memory\Pages Input/sec
  Udává počet stránek, které jsou čtena z disku při řešení chyb spojených se stránkováním. K chybám stránkování dochází, jestliže proces odkazuje na stránku ve virtuální paměti, která se nenachází v pracovní sadě nebo jiné části fyzické paměti a je nutné ji načíst z disku.
- Čtení stránek za sekundu: \Memory\Page Reads/sec
  Udává nám počet stránek, které byly přečteny z disku k řešení chyb stránkování. Zobrazuje počet operací čtení bez ohledu na počet stránek načtených během každé operace.
- Výstupní stránky za sekundu: \Memory\Pages Output/sec
  Udává nám počet stránek zapsaných na disk z důvodu uvolnění místa ve fyzické paměti. Stránky jsou zapsány zpět na disk pouze v případě, že jsou ve fyzické paměti změněny.
- Zápisy stránek za sekundu: \Memory\Page Writes/sec
  Údává počet zápisů stránek na disk z důvodu uvolnění místa ve fyzické paměti.
- Využití stránkovacího souboru [%]: \Paging file(_Total)\% Usage
  Tento čítač nám udává aktuální využití stránkovacího souboru v procentech. Hodnota se může zvýšit zatížením v situaci, kdy operační systém zjistí, že má zálohovat proces s vícestránkovým souborem.

3. Disk

3.1. Propustnost

Propustnost nám udává míru přenosu dat z/do daného disku a většinou se udává v MB/s.

- Čtení z disku [B]: \Physical disk(*)\Disk Read Bytes/sec
  Čítač, který nám ukazuje kolik bylo přeneseno bajtů z disků během čtecích operací za jednu sekundu. Jednoznačně nelze uvést normální hodnotu. Můžeme si však nastavit svoji úroveň běžné hodnoty (základní linii) na základě delšího horizontu pozorování dat. Z této hodnoty pak můžeme vyvozovat, jak je disk vytížen.
- Zápisy na disku [B]: \Physical disk(*)\Disk Write Bytes/sec
  Tento čítač zobrazuje počet přenesených bajtů na disk během čtecích operací za jednu sekundu.

3.2. **Délka fronty**

- Aktuální délka fronty: \Physical disk(*)\Current Disk Queue Length
  Čítač ukazující kolik vstupních/výstupních operací čeká na zapsání/čtení z disku. Jedná se o žádosti, které jsou zasílány na disk a nemohou se ihned zpracovávat. Z toho důvodu jsou posílány do fronty na disku [6].
- Průměrná délka fronty: \Physical disk(*)\Avg. Disk Queue Length
  Tento čítač představuje kolik průměrně vstupně/výstupních operací čeká na zapsání/čtení z disku.

3.3. **Latence a fragmentace**

Latence (prodleva), neboli doba mezi vyvolanou akcí a následkem.
Fragmentace je stav, kdy jsou data na úložišti uložena nesouvisle. Tento vliv způsobuje nižší rychlost při práci s daty a omezení při práci s některými části úložiště.

- Průměrná doba čtení z disku [s]: \Physical disk(*)\Avg. Disk sec/Read
  Tento čítač se váže s ukazatelem prodlevou (latencí) disku. Čím nižší tím lepší. Systém provádí čtení z disku, kde příchozí operace jsou postupně zpracovávány. V průběhu zpracovávání může docházet ke zpoždění. Tuto hodnotu nám ukazuje čítač.
- Průměrná doba zápisu na disk [s]: \Physical disk(*)\Avg. Disk sec/Write
  Čítač, který se rovněž váže k latenci disku. Ukazuje průměrnou dobu, kterou potřebuje ke čtení dat z disku.
- Rozdělení dat na disku: \Physical disk(*)\Split IO/sec
  Čítač, který nám udává míru roztříštění jednotlivých bloků dat do menších bloků dat na disku, neboli do více fragmentů. Tato fragmentace vzniká v důsledku toho, že data jsou např. příliš velká, nejsou na disku uložená v blocích, ale jsou na disku různě roztříštěná do nesouvislých bloků.

4. **Rohraní síťového prvku**

Z pohledu hodnocení výkonnosti rozhraní síťového prvku je zahrnuto pouze kritérium propustnosti. To nám v síti udává množství dat, které je možné v síti přenést za jednotku času.

- Počet přijatých bajtů: \Network Interface(*)\Bytes Received/sec
  Čítač, který ukazuje, kolik daný síťový adaptér přijme bajtů za jednu sekundu (neboli kolik přijme bajtů přes své síťové rozhraní). S ohledem na dnešní rychlosti počítačových sítí je doporučeno převádět výslednou hodnotu na MB/s.
- Počet odeslaných bajtů: \Network Interface(*)\Bytes Sent/sec
  Čítač, který ukazuje kolik daný síťový adaptér odešle bajtů za jednu sekundu (neboli kolik odešle bajtů přes své síťové rozhraní). S ohledem na dnešní rychlosti počítačových sítí je doporučeno převádět výslednou hodnotu na MB/s.

- Délka výstupní fronty: \Network Interface(*)\Output Queue Length
  Ukazatel délky výstupní fronty paketů. Neboli se jedná o množství paketů, které jsou ve frontě a čekají na odeslání.

4.3 Metriky procesů

Proces značí instanci počítačového programu, která je spuštěna v rámci OS. Proces je umístěn v operační paměti v podobě sledu strojových instrukcí, které vykonává procesor počítače. Každý proces má OS přidělené systémové prostředky, prioritu a další vlastnosti jako: ID, stav, název, popis. Každá spuštěná aplikace, software je v operačním systému reprezentována jako proces. Procesy jsou hlavním strůjcem při zatěžování procesoru a operační paměti počítače. Mají tak hlavní vliv na výkonnost a zátěž celého operačního systému. Vybrané metriky pro sledování parametrů procesů jsou:

- Procentuální využití procesoru [%]: \Process(*)\% Processor Time
  Tento čítač ukazuje procentuální využití procesoru daným procesem. Maximální hodnota využití procesoru může být i více než 100 %. Abychom dodrželi standard, kde maximum využití je 100 %, tak byla získaná hodnota přepočítávána podle následujícího vzorce:

$$xp = \frac{v}{p}$$

  kde:

  xp = výsledná hodnota využití procesoru daným procesem (přepočítaná)
  v = získaná hodnota využití procesoru daným procesem (nepřepočítaná)
  p = celkový počet logických procesorů

- Množství vyhrazené (nesdílené) paměti [B]: \Process(*)\Working Set - Private
  Tento čítač nám říká, jaké množství paměti je použito daným procesem v operační paměti. Tato obsazená paměť je soukromá pro daný proces a nemůže být zpřístupněna jinými procesy. Hodnota je v bajtech. Je doporučeno hodnotu převádět do MB/s.

- Vstupně/výstupní data procesu [B]: \Process(*)\IO Data Bytes/sec
  Čítač nám získává množství vstupních/výstupních operací v bajtech. Číslo se odvíjí od míry rychlosti čtení/zápisu vstupně-výstupních operací, které se vykonají za jednu sekundu.

- Počet využívaných vláken: \Process(*)\Thread Count
  Čítač, který ukazuje počet spuštěných aktivních vláken tímto procesem. Každý spuštěný proces obsahuje alespoň jedno vlákno. Z počtu vláken jsme schopni vysledovat, jak je daný proces náročný. Zpravidla jednoduché procesy by si měly vystačit s jedním vláknem. Náročnější pak většinou potřebují více vláken.

- Počet popisovačů (handle): \Process(*)\Handle Count
  Udává nám celkový počet popisovačů, které jsou momentálně otevřené tímto procesem. Popisovač je druh abstrakce, který za sebou ukrývá skutečnou adresu paměti uživatele API a umožňuje systému spravovat fyzickou paměť programu.

4.4 Metriky databází

Získávání informací o databázi je další z klíčových bodů v průběhu monitorování. Databáze obvykle uchovává velké množství dat, o která můžeme lehce přijít, a to z mnoha důvodů jako jsou:

– nezalohování dat,

– přetížení databáze,

– přetečení kapacity.

Z výše popsaných důvodů je zřejmá nutnost sledování několika základních ukazatelů, které vedou k analýze databáze. K získávání informací o výkonu MS SQL Serveru byly použity výkonnostní čítače. Pro pokročilejší a více specifikované informace (tabulky a jednotlivé databáze) byly použity SQL dotazy. Databáze obvykle nabízí několik systémových tabulek, které slouží k získávání informací o následujících skutečnostech:

– vytvořené tabulky,

– cenově N nejnáročnější dotazů s ohledem na CPU,

– využití paměti pro vybranou databázi a tabulku.

Například o tabulce získáme informace o velikosti nepřiřazeného místa a množství využité paměti daty, indexy.

Stejně jako v kapitole 4.2, tak i zde bylo zavedeno jednotné schéma zápisu, které bude platit pro všechny čítače v rámci této kapitoly. Schéma je následující:
„název čítače“: \MSSQL$„název kategorie čítače“\„název konkrétního použitého čítače“ a následuje slovní popis činnosti čítače. Uveďme si vybrané výkonnostní čítače pro MS SQL Server:

- Počet aktuálně připojených uživatelů: \MSSQL$InstanceName:General Statistics:User Connections
  Tento čítač udává hodnotu aktuálně připojených uživatelů k databázi.

- Počet blokovaných procesů: \MSSQL$InstanceName:General Statistics:Processes Block
  Hodnota počtu blokovaných porocesů vzniká, když jeden proces blokuje druhý proces. Zablokovaný proces se nemůže posunout ve frontě (exekučního plánu) dopředu, dokud se blokující proces neuvolní.

- Počet požadavků za sekundu: \MSSQL$InstanceName:SQL Statistics:Batch Requests/sec
  Tento čítač nám ukazuje počet dávek, které se provedou za jednu sekundu. Za jednu dávku se považuje více separátních SQL dotazů, které jsou brány jako skupina příkazů. Druhá možnost je, že se spustí jeden SQL příkaz, který se však skládá z více SQL příkazů. Jedná se o hlavní ukazatel vytížení SQL Serveru.

- Počet kompilací za sekundu: \MSSQL$InstanceName:SQL Statistics:SQL Compilations/-sec
  Tento čítač nám udává, kolikrát SQL Server vytvoří (zkompiluje) exekuční plán za sekundu. Získané číslo z čítače by mělo být porovnáno s čítačem počet požadavků za sekundu, aby se zjistilo, zda každá kompilace snižuje výkonnost. Obecně platí, že počet kompilací za sekundu by měl být 10 % a méně než je celkový počet požadavků za sekundu.

- Počet znovuzkompilování za sekundu: \MSSQL$InstanceName:SQL Statistics:SQL Re-Compilations/sec
  Čítač nám udává hodnotu na základě toho, kolikrát nastane nějaká významná událost, která zapřičiní, že SQL Server nevytvoří exekuční plán.

- Hodnota doby čtení z paměti: \MSSQL$InstanceName:Buffer Manager:Buffer cache hit ratio
  Tento čítač nám udává, jak často SQL Server dokáže najít data v zásobníku mezipaměti a to ve chvíli, kdy dotaz potřebuje patřičná data stránky. Rozmezí hodnoty je 0-100, přičemž 100 znamená, že po celou dobu hledání měl vždy potřebná data ihned v paměti.

- Očekávaná délka doby stránky: \MSSQL$InstanceName:Buffer Manager:Page life expectancy
  Čítač, který udává, po jak dlouhou dobu (v sekundách) jsou stránky v zásobníku mezipaměti. Čím delší doba je, tím větší je pravděpodobnost, že data stránky se budou číst z paměti a ne z disku.

- Líné zápisy za sekundu: \MSSQL$InstanceName:Buffer Manager:Lazy Writes/sec
  Čítač, který nám ukáže, kolikrát došlo k přesouvání špinavých stránek ze zásobníku na disk tak, aby se uvolnilo místo (zásobníku). Stránky z mezipaměti, které byly v průběhu změněny se nazývají špinavé stránky.

- Počet zámků za sekundu: \MSSQL$InstanceName:Locks(_Total):Lock Waits/sec
  Aby mohl SQL Server spravovat souběžné uživatele v systému, musí čas od času uzamknout zdroje. Čítač udává hodnotu na základě toho, kolikrát SQL Server za sekundu nedokáže udržet zámek pro zdroj.

- Rozdělování stránek za sekundu: \MSSQL$InstanceName:Access Methods:Page Splits/sec
  Čítač počtu rozdělení stránek, které nastanou během aktualizace nebo vkládání dat za sekundu. Rozdělování stránek obvykle nastane, pokud na stránce nezůstane žádný další prostor

pro vkládání nebo aktualizování dat. SQL Server pak přenáší data z této zaplněné stránky do jiné datové stránky. Rozdělování stránek jsou cenově drahé operace, takže to může mít rovněž vliv na výkon. Hodnota by se měla pohybovat mezi 10-20 % z celkového počtu požadavků za sekundu.

K získávání detailnějších informací byly použity SQL dotazy, které se zaměřovaly na dotazování systémových dat SQL Serveru. Tato data jsou zpravidla uložená v tabulkách, které začínají názvem sys. Informace získané o konkrétní databázi, tabulce a dotazu jsou ukládány do mezipaměti. To znamená, že pokud SQL Server běží, tak kontinuálně přibývají informace o vybraných periferiích. Z toho však plyne, že pokud informace nejsou zalohovány nebo ukládány, tak při vypnutí či restartování SQL Serveru přijdeme o všechny tyto informace.

4.4.1 Query Store

SQL Server nabízí sledování konkrétních dotazů a jejich parametrů za pomocí Query Store. Jedná se o nástroj, který slouží ke sledování dotazů, exekučních plánů, vývoje a výkonnostních statistik. Usnadňuje monitorování výkonu a řešení výkonnostních potíží spojených pouze s SQL dotazy. Nejedná se o nástroj, který by nám umožnil sledovat samotnou zátěž a výkonnost databáze, tabulek, případně celý SQL Server. Slouží výhradně pro sledování periferií spojených s dotazy. Tento nástroj je dostupný až od verze SQL Server 2016.

Pro starší verze SQL Serveru je zapotřebí napsání vlastních dotazů, které se budou dotazovat na patřičné systémové tabulky. Můžeme si uvést, které tabulky budeme používat při dotazování na SQL Server, a také si uvedeme několik konkrétních příkladů SQL dotazů.

- Systémová tabulka sys.dm_exec_query_stats
  Tabulka nám umožňuje získávát souhrnné informace o dotazech. Informace se uchovávají v mezipaměti SQL Serveru. Tyto informace jsou v mezipaměti uloženy po omezenou dobu. Při smazání mezipaměti serveru o tato data přicházíme.

- Systémová tabulka sys.dm_exec_procedure_stats
  Tabulka, která umožňuje získávát informace o procedurách, které jsou uložené na straně SQL Serveru.

- Systémová tabulka sys.dm_exec_sql_text
  Tato tabulka nám vrací dávku SQL, tabulka přijímá jako parametr „sql_handle“ nebo „plan_handle“.

- Systémová tabulka sys.databases
  Tabulka vrací infromace o všech databázích v rámci SQL Serveru. Jeden řádek představuje informace o dané databázi.

- Tabulka INFORMATION_SCHEMA.COLUMNS
  Tabulka, která nám umožní získat informace o struktuře tabulky. Dále informace jako jsou

katalog, schéma, apod. Je však nutné při dotazování uvést do klauzule „WHERE“ název tabulky, o které chceme data získávat. V opačném případě získáme informace pro všechny tabulky v právě využívané databázi.

- Systémová procedura sp_spaceused
  Tato procedura nám umožňuje získávat informace o využití databázového souboru až po velikost vybrané tabulky. Například počet záznamů tabulky, rezervované místo na disku, využité místo na disku tabulkou.

```
SELECT TOP 10 total_worker_time/execution_count AS Avg_CPU_Time,
    execution_count, total_elapsed_time/execution_count as AVG_Run_Time,
(
    SELECT SUBSTRING(text,statement_start_offset/2,
    (
        CASE WHEN statement_end_offset = -1 THEN
            LEN(CONVERT(nvarchar(max), text)) * 2 ELSE
            statement_end_offset END
            -statement_start_offset)/2
    ) FROM sys.dm_exec_sql_text(sql_handle)
) AS query_text
FROM sys.dm_exec_query_stats
ORDER BY Avg_CPU_Time DESC
--ORDER BY AVG_Run_Time DESC
--ORDER BY execution_count DESC
```

Výpis 1: Nejhorších deset výkonnostních dotazů podle průměrného využití CPU

```
--use <Instance database_name> ;
IF OBJECT_ID('tempdb..#SpaceUsedOfTable') IS NOT NULL DROP TABLE #
    SpaceUsedOfTable;
GO
CREATE TABLE #SpaceUsedOfTable
(
  TableName sysname
  ,NumRows BIGINT
  ,ReservedSpace VARCHAR(50)
  ,DataSpace VARCHAR(50)
  ,IndexSize VARCHAR(50)
  ,UnusedSpace VARCHAR(50)
);
```

```
INSERT INTO #SpaceUsedOfTable
EXEC sp_spaceused '[table name]'; --replace table name to your table

SELECT TableName, NumRows,
CONVERT(numeric(18,0),REPLACE(ReservedSpace,' KB','')) AS ReservedSpace_KB,
CONVERT(numeric(18,0),REPLACE(DataSpace,' KB','')) AS DataSpace_KB,
CONVERT(numeric(18,0),REPLACE(IndexSize,' KB','')) AS IndexSpace_KB,
CONVERT(numeric(18,0),REPLACE(UnusedSpace,' KB','')) AS UnusedSpace_KB
FROM #SpaceUsedOfTable
ORDER BY ReservedSpace_KB DESC;
```

Výpis 2: Využití konkrétní tabulky pro danou databázi

4.5 Architektura WMI

Jedná se o infrastrukturu pro správu dat a operací v OS Windows. Poskytuje jednotné rozhraní pro získávání dat, správu počítačového systému a sítě. Data je možné získávat jak z lokálního počítače, tak ze vzdáleného počítače. Je však zapotřebí mít nastavené patřičná oprávnění (uživatel musí být přiřazen do patřičných skupin, musí mít oprávnění typu administrátor a nastavený „Windows Firewall“). WMI je nainstalováno na všech počítačích, které jsou ve verzích: Windows 2000, Windows XP, Windows Server 2003 a vyšší.

Primární využití WMI bude založeno na používání WMI tříd. Tyto třídy jsou kolekcí z předdefinovaných tříd založených na CIM. Třídy, které byly použité, tak spadaly většinou do kategorie hardwarových nebo operačního systému.

Uveďme si některé třídy, které byly použité: „Win32_VideoController“, „Win32_Process“, , „Win32_DiskDrive“, „Win32_OperatingSystem“. Abychom však mohli získávat data o těchto třídách, tak bylo zapotřebí využít další třídy. Ty nám zprostředkovaly informace, které chceme vyčíst. Třída, která nám umožnila napsat dotaz na požadované informace, byla „ObjectQuery“. Další využitou třídou byla „ManagementObjectSearcher“, která nám umožnila data získat do instance třídy „ManagementObject“. Z tohoto objektu jsme pak mohli získat potřebné informace na základě vlastností, které třída obsahovala.

V případě, že se připojujeme na pracovní stanici vzdáleně, tak je potřeba navíc využít třídu „ManagementScope“, která nám zajistí vzdálené připojení.

5 Řešení problematiky logování a výkonnostních metrik

Výsledná aplikace byla vytvořena pro monitorování a získávání informací o čtyřech oblastech, viz. kapitola 4. Před implementací aplikace bylo potřeba vytvořit koncept logování. Logování jsme rozdělili na dvě části. V první části jsme potřebovali využít logování pro získávání informací o naší aplikaci. Vytvořili jsme jednotný standard, který byl následně aplikován v rámci celé aplikace. V této oblasti byly využité technologie log4net a RabbitMQ.

V druhé části se jednalo o vytváření tzv. analytického logu. Aplikace byla koncipována tak, aby bylo možné získávat logy z lokálních, ale také vzdálených počítačů. To znamená, že při logování vzdálených počítačů/serverů jsme předávali při vytváření instance třídy do parametru instance přihlašovací údaje. To probíhalo za využití třídy „Credentials“. U lokálních počítačů jsme pak vytvářeli instanci třídy bez parametru.

V oblastech monitorování výkonu operačního systému, procesů a databází byly použity primárně výkonnostní čítače. Oblast monitorování databáze byla rozdělena na dvě části. V prvním případě se jednalo o monitorování výkonu relačního systému MS SQL Server. Ve druhém případě se jednalo o monitorování konkrétních databázových instancí a tabulek, které jsou vytvořené v rámci systému. Pro získávání informací o tabulkách a konkrétních databázích byly vytvořeny SQL dotazy, díky kterým jsme získavali informace pro detailnější analýzu. Poslední oblastí bylo získávání Windows událostních logů. Tyto logy byly získávány za pomocí implementace jak již dostupných tříd, tak rovněž za pomocí implementací vlastních nových tříd, které poskytovaly potřebné metody pro prácí s logy.

5.1 Zavedení standardu a řešení logování aplikace

Prvním krokem, který vedl k nastavení logovacího mechanismu, bylo vybrání vlastností, které měl log obsahovat tak, aby výsledné informace vyhovovaly potřebám pro analýzu systému. Dalším krokem bylo si určit základní linii, která určovala, jaké informace budeme logovat. Následně byla nastavena jednotná úroveň ukládání informací (sjednocení úrovní logů), která nám zajistila kategorizaci logů podle druhu informace. Poslední fází bylo určení úložiště a formát ukládání dat.

Technologie log4net se skládá ze 3 komponent, které jsou: „logger“, „appender“ a „layout“. Flexibilita log4net umožnila, že můžeme současně ukládat informace do textového souboru, který byl umístěn lokálně na disku, ale také do databáze Elasticsearch. Je však nutné provést patřičné nastavení v souboru „App.config“. Pro větší nezávislost na výběru OOP jazyka a větší flexibilitu při použití logovacího konceptu jsme využili technologii RabbitMQ. Tato technologie vstupovala jako prostředník mezi log4net a Elasticsearch databází.

Často využívanou technikou při logování informací bylo využívání „try/catch“ bloků. Kde v „try“ těle byla požadovaná část kódu, který se měl provést. A následně v těle „catch“ bloku jsme zalogovali informace, které jsme získali v případě neúspěšného provedení kódu. Rovněž bylo nutné určit si, s jakou úrovní se má informace zalogovat. Úroveň, se kterou je log uložen,

si však musí každý určit a to na základě vytvořeného standardu. Kroky, které vedly k implemetaci a použití logovacího nástroje (log4net), který bude logy ukládat přímo do Elasticsearch databáze, si můžeme ukázat v následujících bodech.

1. Implementace třídy „Log“, která obsahovala patřičné informace.
2. Implementace třídy „LogLayout“, která specifikuje výstupní formát.
3. Implementace třídy „ElasticsearchAppender“, která odešle formátovaný výstup do Elasticsearch databáze.
4. Přidání a natavení tagu „<log4net>“ v souboru „App.config“.
5. Vytvoření instance „loggera“ ve třídě, v rámci které chceme logovat.
6. Použití instace „loggera“ s využitím jeho metod pro zaznamenání informace.

Výsledný log byl reprezentován těmito vlastnostmi:
zpráva, výjimka, jméno logeru, doména, totožnost, úroveň (číslo), úroveň (slovně), název třídy, název souboru, číslo řádku, celkové informace, oprava, název metody, uživatelské jméno, jméno počítače, verze, zdroj, IP počítače a časové razítko.

Obrázek 1: Schéma logování informací

Při řešení standardu logování aplikace jsem vytvářel:

– „appender“ pro Elasticsearch a RabbitMQ,
– třídy pro repretentaci logů.

5.2 Řešení pro sledování událostních logů

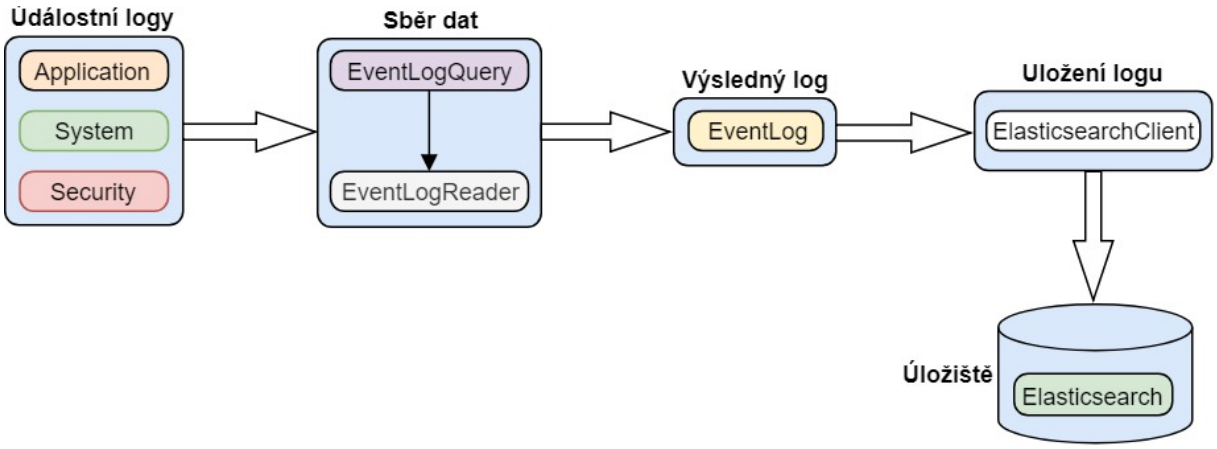
Operační systém Windows generuje událostní logy, ze kterých lze získávat informace o běhu systému. Tyto logy jsou ukládány do souborů datového typu „Protokol událostí“ (.evtx). Soubory najdeme ve složce „Logs“, která spadá do systémových složek Windows. Náhled na obsah těchto souborů nám umožňovala aplikace eventvwr.exe, viz. 4.1.1. Logy jsou škálované do souborů na základě typu události, o kterou v systému šlo. V aplikaci jsme získavali logy z kategorie „Protokoly systému Windows“. Z této kategorie jsme vybrali aplikační a systémové logy a dále logy týkající se zabezpečení systému.

Vyčítání logů probíhalo jak z lokálního počítače, tak ze vzdáleného. U lokálního bylo potřeba oprávnění uživatele typu administrátor. Ze vzáleného počítače musí být uživatel přiřazen

do skupiny „Event Log Readers“ a rovněž mít oprávnění uživatele typu administrátor. Generování logů v systému je založeno na základě vyvolané události, která je následně zaznamenána. Z tohoto důvodu bylo vyčítání logů naimplementováno asynchronně.

U vyčítání logů z operačního systému Windows byly primárně využity třídy „EventLogQuery“ a „EventLogReader“ [7]. Třída „EventLogQuery“ nám umožnila se dotazovat na logy z vybraných kategorií a následně je přečíst za pomocí třídy „EventLogReader“. Před uložením do výsledného logu bylo potřeba sjednotit, o jaký typ informace se jedná. To bylo dosaženo na základě úrovně logu a klíčových slov obsažených ve vygenerovanám logu. Pro získání detailní zprávy o logu bylo nutné přečtený log převést a následně uložit vybrané informace.

Výsledný log byl reprezentován třídou „EventLog“, která měla následující vlastnosti:
ID události, detailní zpráva s podrobnými infromacemi, úroveň (číslo), úroveň (slovně), kategorii úlohy, název logu (druh), uživatelské jméno, typ, zdroj, jméno počítače, IP počítače a časové razítko.



Obrázek 2: Schéma získávání událostních logů

Při řešení získávání Windows událostních logů jsem vytvářel:

– třídy pro sběr dat,

– třídu pro reprezentaci logu,

– asynchronní koncept ukládání logů.

Implementaci pro vyčítání Windows událostních logů můžeme najít v příloze B.1. Získávání podrobnějších informací, které vedly k analýze činnosti systému pak najdeme v příloze B.2.

5.3 Řešení pro sledování výkonu operačního systému

Výkon operačního systému je zpravidla úměrný tomu, jaký máme použitý hardware na konkétním počítači. Proto byl použit koncept, který je rozškálován na základě hardwarových komponent. Jednalo se o rozdělení podle procesoru, fyzické paměti (RAM), pevného disku a síťového rozhraní. Tyto vybrané komponenty byly klíčové pro sledování výkonu.

K získávání informací o komponentách, byly použity výkonnostní čítače, které jsou součástí operačního systému. Tyto čítače jsou rozdělené do kategorií, které vyhovují našemu zvolenému konceptu rozdělení. K použití těchto čítačů na lokálním počítači je potřeba uživatelského oprávnění administrátor. U získávání dat ze vzdáleného počítače je navíc potřeba být ve skupině „Performance Monitor Users“. Získávání informací se provádělo v opakujících se intervalech, které lze modifikovat na základě potřeby. Náš zvolený interval byl 5 minut.

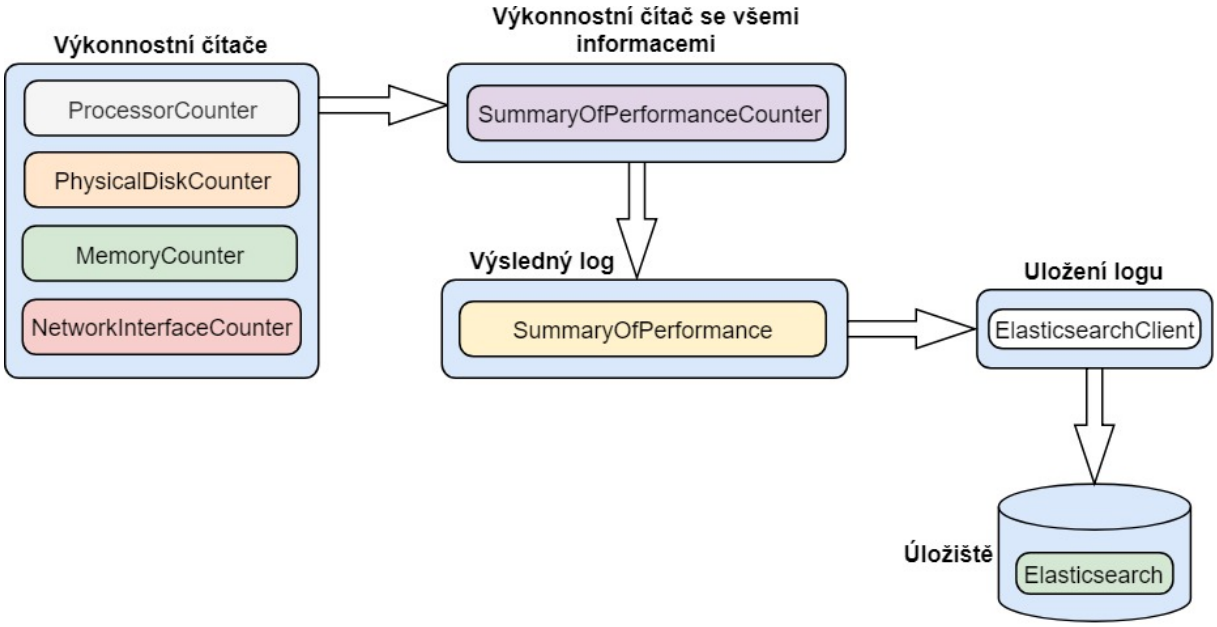
K získávání hodnot výkonu pro jednotlivé komponenty, sloužila třída „Performance Counter“. Tato třída byla klíčová a nabízela získávání hodnot jak lokálně, tak i vzdáleně. K vyčítání hodnot ze vzdáleného počítače, bylo potřeba mít inicializaci čítače v definici těla instance třídy „Impersonator“. Tato třída nám umožnila, že jsme schopni vzdáleně získávat informace s použitím třídy „Performance Counter“.

V implementační části jsme vytvořili třídy, které nám zajistily získávání hodnot a jejich následnou reprezentaci dat. Vytvářely se vždy dvojice tříd, kde první byla např. „ProcessorCounter“ pro získání všech požadovaných informací a následně reprezetace dat do třídy „Processor“. Tato konkrétní třída pak měla vlastnosti:

- celkové využití procesoru,
- využití jednotlivých procesorů (byl použit datový typ „dynamic“),
- celkové využití procesoru v privilegovaném režimu,
- celkové využití procesoru v uživatelském režimu.

Všechny tyto vlastnosti představovaly procentuální využití, kde maximum bylo 100 %.

Po vytvoření všech těchto tříd byla vytvořena konečná třída, která obsahovala všechny vytvořené třídy. Výsledná třída nabízí komplexní informace o všech potřebných komponentách, které jsou potřebné pro analýzu o operačním systému. Třída byla ještě doplněna o vlastnosti: jméno počítače, IP počítače a časové razítko, kdy byl log vytvořen. V momentě, kdy byl tento objekt naplněný daty, tak bylo možné objekt uložit do databáze Elasticsearch, viz. kapitola 5.6.



Obrázek 3: Schéma získávání informací o výkonu operačního systému

Při řešení sledování výkonu OS jsem vytvářel:

– třídy pro sběr dat,

– třídy pro reprezentaci logů.

Implementace výkonnostních čítačů, které sloužily pro ohodnocení výkonu procesoru můžeme najít v příloze A.1 a A.2.

5.4 Řešení pro sledování o zatížení procesy

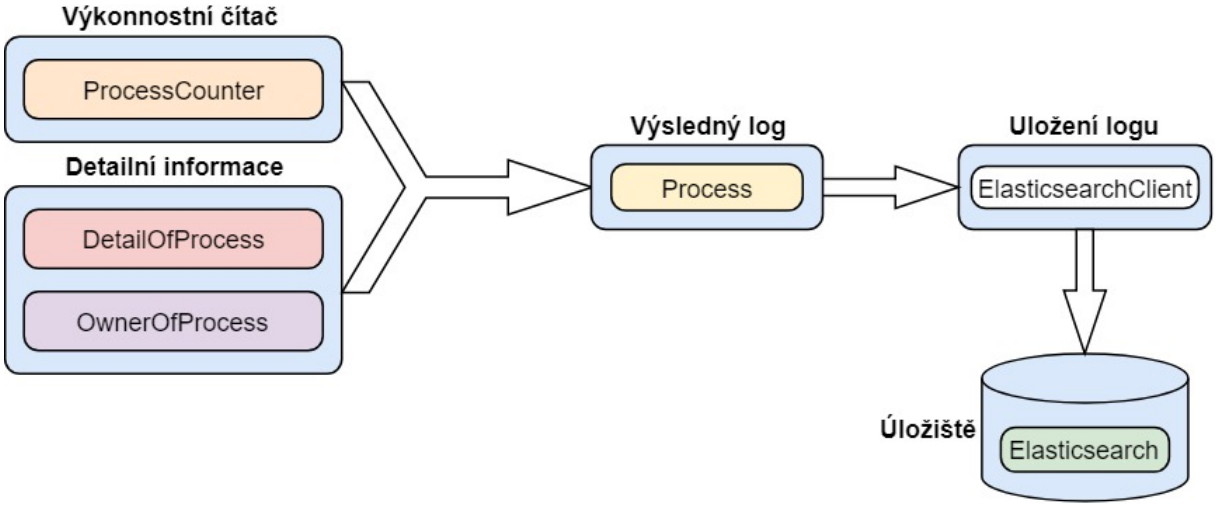
Sledování výkonů procesů je nepřehledné a náročné. Hlavním problémem při implementaci pro monitorování výkonu procesů byl fakt, že procesy jsou za běhu často ukončovány a vytvářeny. Bylo tak nutné implementované třídy koncipovat a optimalizovat tak, aby si s tímto faktorem poradily. U tohoto faktoru byl využitý náš nastavený mechanismus pro logování. Logování nám zajistilo, že pokud v průběhu monitorování daného procesu byl proces ukončen, tak aplikace nespadla. Ale naopak byla informace úspěšně zalogována.

Při získávání informací o výkonu procesů jsme primárně využili třídu „PerformanceCounter“. Princip nastavení patřičného oprávnění při získávání hodnot ze vzdálené pracovní stanice je stejný jako v minulé kapitole viz. kapitola 5.3. Byly vytvořeny dvě hlavní třídy. První sloužila pro sběr informací o procesech a druhá sloužila pro reprezentaci dat. První zmíněná se jmenovala „ProcessCounter“ a druhá pak „Process“.

Kromě využití třídy „PerformanceCounter“ byla využita i diagnostika prostřednictvým rozhraní WMI, viz. kapitola 4.5. Za pomoci tohoto rozhraní jsme získali informace o cestě,

ze které jsou jednotlivé procesy spouštěny. Dále jsme ještě získali informace o uživateli, pod kterým je proces spuštěn. Cesta, která byla získána za pomoci WMI, byla nutná pro předání třídě „FileVersionInfo“, která poskytla informace o popisu procesu, verzi a společnosti.

Výsledný log o procesu měl následující vlastnosti:
ID procesu, jméno, celkové využití procesorem, počet obsluhovačů (handle), počet využívaných vláken, vstupně-výstupní operace (čtení/zápis), přiřazenou fyzickou paměť (soukromá), jméno uživatele, popis, verze, společnost, doba po jakou je proces spuštěn, jméno počítače (na kterém proces běží), IP počítače (na kterém proces běží) a časové razítko. Následně byl takto naplněný objekt uložen do databáze Elasticsearch, viz. kapitola 5.6.



Obrázek 4: Schéma získávání informací o výkonu procesů systému

Při řešení zatížení výkonu OS procesy jsem vytvářel:

- třídy pro sběr dat,
- třídu pro reprezentaci logů,
- Asynchronní koncept ukládání logů.

5.5 Řešení pro sledování výkonu databáze

Sledování databáze bylo rozděleno na dvě části. První část byla zaměřena na získávání informací o vytížení databázového systému MS SQL Server. K získání těchto informací byly použity výkonnostní čítače. Pro komplexní pozorování vytížení byly sledovány následující oblasti:

- počet a frekvence spouštěných SQL příkazů,
- vytížení paměti,
- počet aktuálně připojených klientů,

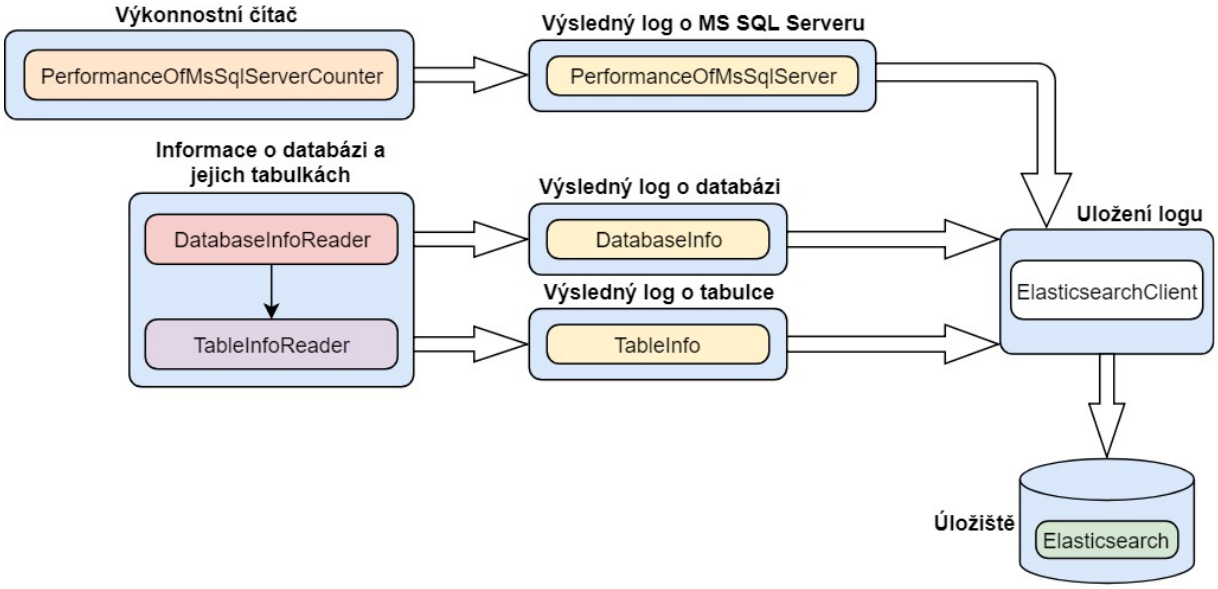
– opoždění (prodleva) při čtení a zápisu dat.

Vznikaly vždy dvojice tříd, kde první se jmenovala např. „BufferManagerCounter“ a sloužila pro sběr informací. Druhá pak „BufferManager“ a sloužila pro reprezentaci získaných dat. Tímto postupem vznikaly další potřebné třídy, které získaly všechny námi vybrané informace z oblasti sledování vytížení. Mohla tak vzniknout výsledná třída „PerformanceOfMsSqlServerCounter“, která obsahovala komplexní informace o výkonu MS SQL Serveru.

Druhá část sledování byla zaměřená na získávání informací o konkrétních instancích databází a jejich tabulkách. Pro tyto informace byly naimplementovány třídy „DatabaseInfoReader“ a „TableInfoReader“. V obou třídách byly naimplementovány metody, které obsahovaly SQL dotazy, které získaly námi vybrané informace.

Pro reprezentaci informací o databázích vznikla třída „DatabaseInfo“, která obsahovala: jméno, logický název, velikost v MB, počet tabulek v rámci dané databáze, status, možnost aktualizace, typ zálohování, využité místo v MB, využité místo v %, poslední datum zálohy, typ poslední zálohy, velikost poslední zálohy, jméno databázového systému, IP počítače na kterém databázový systém běží a časové razítko, kdy byla informace vytvořena.

Pro reprezentaci informací o tabulkách vznikla třída „TableInfo“, která obsahovala: jméno, počet záznamů, velikost v MB, využité místo v MB, využité místo %, velikost indexu v MB, jméno databáze do které tabulka patří, systému, jméno databázového systému, IP počítače na kterém databázový systém běží a časové razítko, kdy byla informace vytvořena.



Obrázek 5: Schéma získávání informací o výkonu databází

Při řešení sledování výkonu o databázi jsem vytvářel:

– třídy pro sběr dat o vytížení MS SQL Serveru,

– třídy pro reprezentaci logů o vytížení MS SQL Serveru,

– třídy pro sběr dat o využití konkrétních databází,

– třídy pro reprezentaci logů o využití konkrétních databází,

– třídy pro sběr dat o využití konkrétních tabulek v rámci dané databáze,

– třídy pro reprezentaci logů o využití konkrétních tabulek v rámci dané databáze,

– koncept ukládání získaných logů.

5.6 Koncept ukládání dat a jejich vizualizace

Převážná většina dat byla uložena v NoSQL databázi Elasticsearch. Zbývající data byla serializována do JSON struktury a následně uložena do souboru typu JSON (.json). Výhody a nevýhody těchto dvou typů úložišť si můžeme shrnout v následujících odrážkách (výhody jsou reprezentovány symbolem + a nevýhody symbolem –).

1. Ukládání do souboru
   + Jednoduchá implementace
   + Jednoduché zálohování
   + Rychlost zápisu, čtení (platí pro menší soubory, s přibývajícími daty se rychlost čtení snižuje)
   – Nízká úroveň zabezpečení (pokud není ukládán na cloud)
   – Bez možnosti dotazování
   – Formát (nevhodný, vzhledem k použité technologii), struktura dat (špatná čitelnost)
2. Ukládání do databáze NoSQL
   + Centralizace dat
   + Dotazování dat (flexibilita při získávání dat na základě výběru požadovaných vlastností a podmínek)
   + Formát, struktura dat
   + Bezpečnost
   + Bez potřeby návrhu datového modelu (při ukládání dat)
   – Zálohování dat (nutná pokročilejší znalost, konfigurace)
   – Administrace (přidělování oprávnění uživatelům k uloženým datům)

Stejně jako u výhod, které jsou uvedené pro NoSQL databáze, tak i u Elasticsearch databáze platí, že není potřeba definovat strukturu (je vytvořena na základě vlastností objektu). Tato vlastnost NoSQL databází byla klíčová, jelikož výsledný log měl všechny potřebné informace

pro následnou analýzu dat. Uložená data v databázi jsou ukládána v JSON formátu. Díky této vlastnosti jsme mohli data ze souboru ukládat i do databáze. Tato možnost přinesla větší flexibilitu ukládání dat.

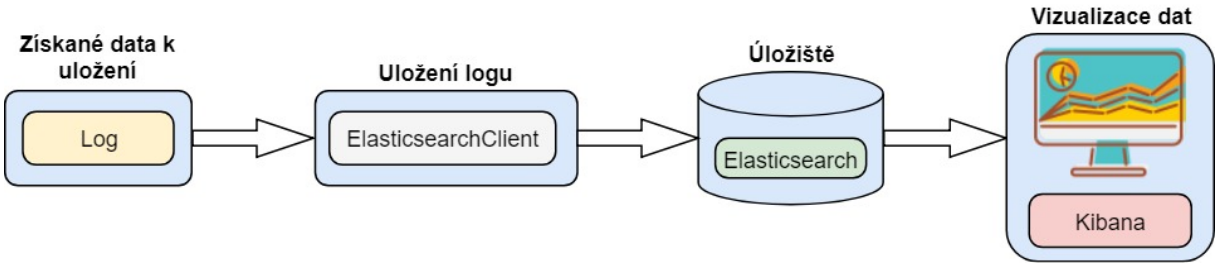
Pro ukládání dat do databáze byla vytvořena třída, která reprezentuje potřebné vlastnosti a metody pro práci s Elasticsearch databází. Nejpodstatnější vlastnosti byly:

- jméno hosta,
- jméno portu,
- jméno indexu (pod kterým budou data mapována).

Třída byla generická, takže bylo možné při inicializaci instance třídy definovat jakýkoliv datový typ. Genericita třídy umožnila uložení jakéhokoliv objektu do Elasticsearch databáze. Elasticsearch funguje na pricipu ukládání, kde při uložení objektu je nutné definovat pod jakým indexem (unikátním názvem) má být příslušný objekt uložen. Tento princip je pak využit při mapování na objekty, tzn. abychom získali celou kolekci dat z příslušného indexu.

Pro vizualizaci dat byl použit vizualizační nástroj Kibana. Nástroj je dostupný přes webový prohlížeč. V našem případě byl dostupný v rámci interní sítě ArcelorMittal. V nástroji Kibana bylo prvním krokem k prohlížení dat vytvoření indexu, který bude odpovídat názvu indexu z Elasticsearch databáze. Tato nalezená shoda následně zajistí, že jsme schopni pozorovat kolekci dat pro vybraný index z Elasticsearch databáze. V posledním kroku je potřeba vybrat, na základě které vlastnosti objektu se má vytvořit časová osa. Tato časová osa slouží pro reprezentaci časového průběhu a analýzu dat. Jakmile byl vytvořen index, bylo možné data prohlížet, provádět agregace a nastavit filtry. Kibana nabízí možnost vytváření vizualizačních oken, která se skládají z vizualizačních komponent. Díky těmto komponentám jsme byli schopni získaná data vizualizovat. Při tvorbě vizualizačních oken je nabídka velké škály vizualizačních možností. Ve většině případů byly použity vizualizační komponenty typu „Visual Builder“. Například pro vizualizaci dat o zatížení procesoru byl vytvořen čárový graf se třemi křivkami. První reprezentovala celkové využití procesoru. Druhá celkové využití procesoru v privilegovaném režimu. Poslední pak využití procesoru v uživatelském režimu. Každá křivka byla vykreslena na základě dat, která byla získáná z Elasticsearch databáze. Tato data byla pro každou křivku agregována podle průměrné hodnoty vlastnosti objektu, např. „processor.totalUtilization“.

Tímto postupem byly vytvořeny všechny vizualizační komponenty. Jakmile byly tyto komponenty sestaveny, byla vytvořena vizualizační okna. Vizualizační okna nabízela časový rozsah, ve kterém je možné data pozorovat. V našem případě byly data zobrazeny posledních 24 hodin.



Obrázek 6: Schéma ukládání získaných dat a následná vizualizace

Při řešení konceptu ukládání dat a vizualizace jsem vyvářel:

– koncept ukládání dat,

– třídu pro ukládání dat do Elasticsearch databáze,

– třídy pro ukládání a práci se soubory,

– vizualizační komponenty v nástroji Kibana,

– vizualizační okna v nástroji Kibana.

Implementace vybraných metod, které slouží pro ukládání výsledného logu do Elasticsearch databáze můžeme najít v příloze C.

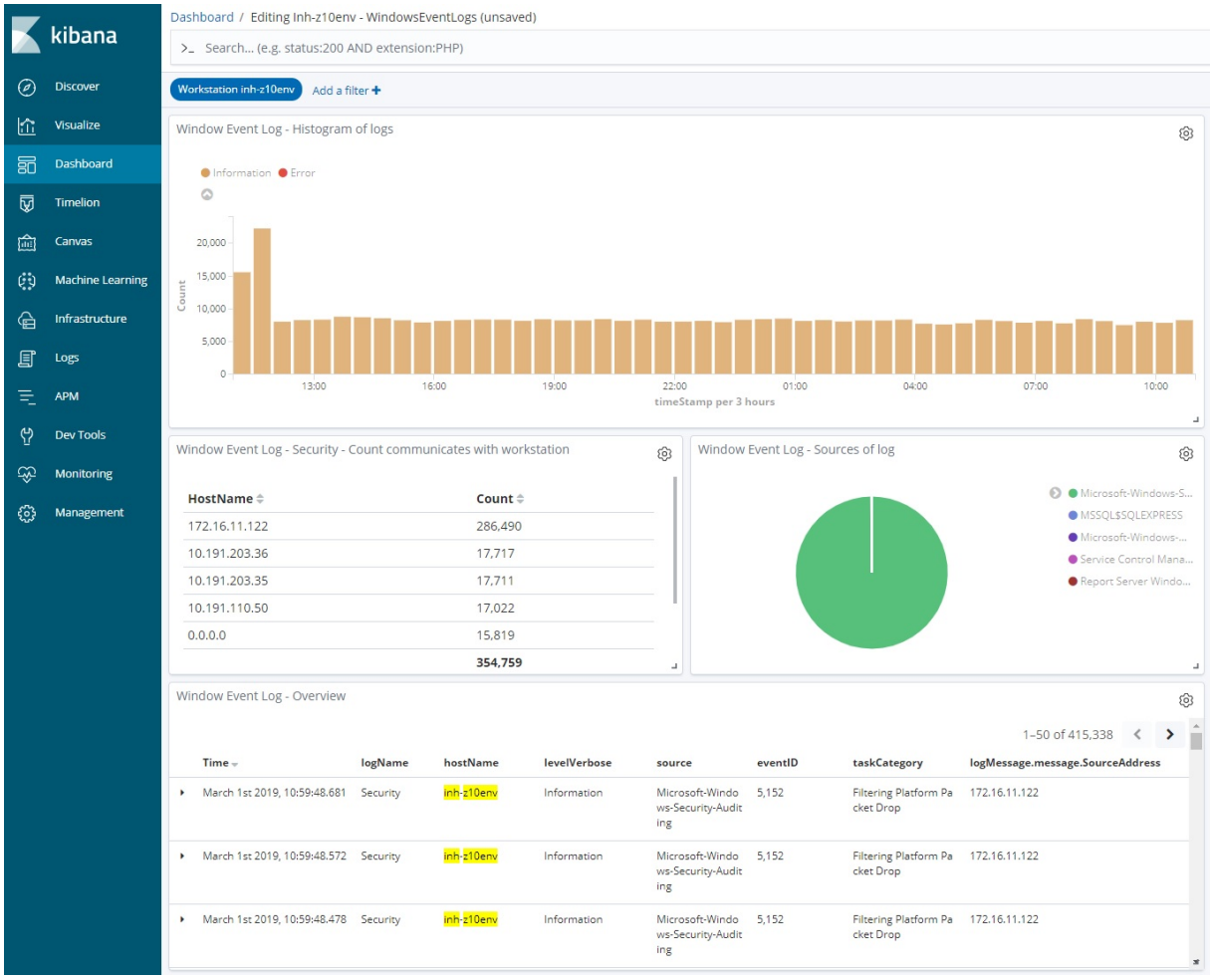
5.7 Vizualizace výsledných dat

V prostředí Kibana byla vytvořena vizualizační okna, která reprezentují námi vybrané oblasti, viz. kapitoly 4.1 - 4.4. Vizualizační okna reprezentují data z počítačů, které byly monitorovány.

5.7.1 Získané událostní logy

Obrázek 7 ukazuje vybrané vizualizační komponenty, které nám poskytují nejdůležitější informace o Windows událostních lozích v reálném čase. Na základě těchto dat jsme detekovali:

- stabilitu systému na základě počtu chyb,
- počet komunikací (hostujícího počítače) s vybraným sledovaným počítačem,
- podrobné informace o běhu systému.



Obrázek 7: Vygenerované Windows událostní logy v průběhu jednoho dne

Analýza událostních logů nám zajistila informace o běhu systému, aplikací a zabezpečení. Díky těmto informacím jsme mohli v reálném čase reagovat na vzniklé chyby a následně je řešit.

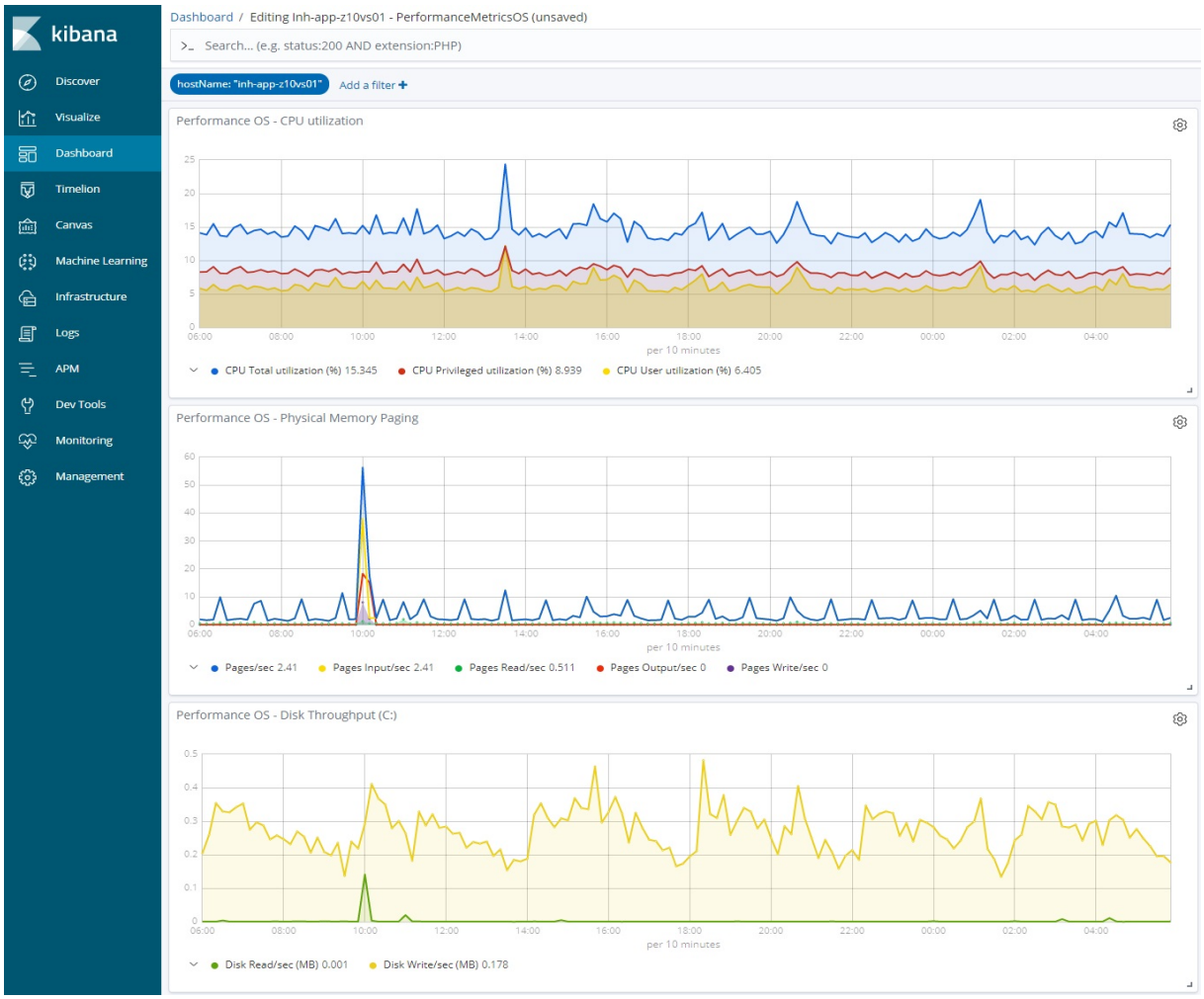
5.7.2 Metriky operačního systému

Obrázek 8 obsahuje vybrané metriky, které demonstrují vytížení OS v reálném čase v průběhu jednoho dne. Na základě těchto dat jsme detekovali, že v systému dochází k následujícím událostem:

– spouštění procesů v pravidelných intervalech,

– aktulizacím,

– zálohování databází.



Obrázek 8: Průběh činnosti procesoru, fyzické paměti a disku za jeden den

Prostřední graf na obr. 8 demonstruje zvýšenou míru čtení dat z pevného disku do operační paměti v 10:00 hodin. Tato hodnota indikovala možné chyby při čtení stránkovacího souboru z pevného disku. Zásluhou monitorování vytížení OS bylo možné optimalizovat OS nebo dokonce vyměnit či rozšířit nedostačující hardwarové komponenty.

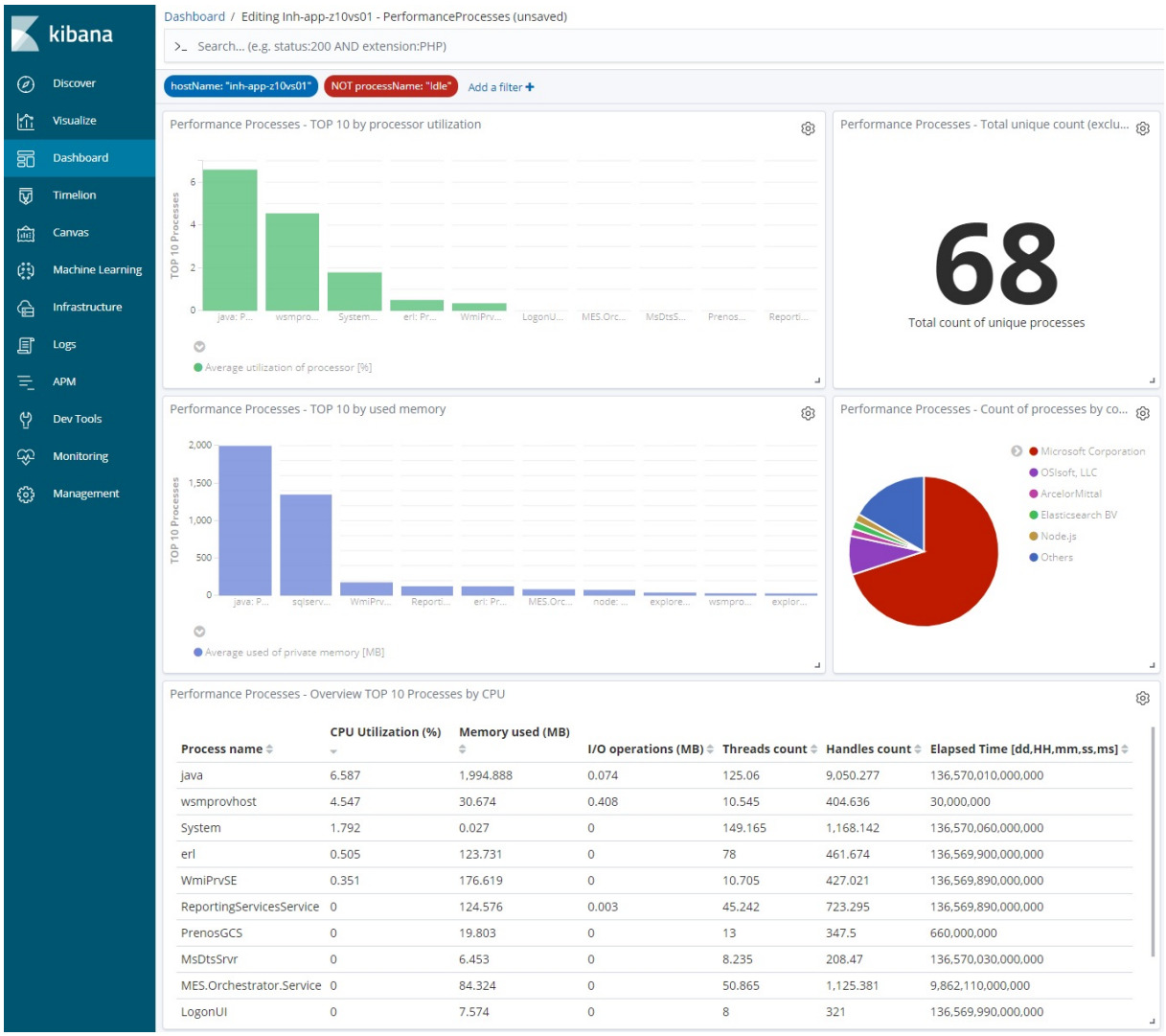
5.7.3 Metriky procesů

Obrázek 9 obsahuje vybrané vizualizační komponenty, které demonstrují vytížení OS procesy v reálném čase. Na základě dat z procesů jsme mohli detekovat:

– procesy, které nejvíce zatěžují běh OS s ohledem na využití procesoru,

– procesy, které nejvíce zatěžují běh OS s ohledem na využití fyzické paměti,

– počet běžících procesů v operačním systému.



Process name	CPU Utilization (%)	Memory used (MB)	I/O operations (MB)	Threads count	Handles count	Elapsed Time [dd,HH,mm,ss,ms]
java	6.587	1,994.888	0.074	125.06	9,050.277	136,570,010,000,000
wsmprovhost	4.547	30.674	0.408	10.545	404.636	30,000,000
System	1.792	0.027	0	149.165	1,168.142	136,570,060,000,000
erl	0.505	123.731	0	78	461.674	136,569,900,000,000
WmiPrvSE	0.351	176.619	0	10.705	427.021	136,569,890,000,000
ReportingServicesService	0	124.576	0.003	45.242	723.295	136,569,890,000,000
PrenosGCS	0	19.803	0	13	347.5	660,000,000
MsDtsSrvr	0	6.453	0	8.235	208.47	136,570,030,000,000
MES.Orchestrator.Service	0	84.324	0	50.865	1,125.381	9,862,110,000,000
LogonUI	0	7.574	0	8	321	136,569,990,000,000

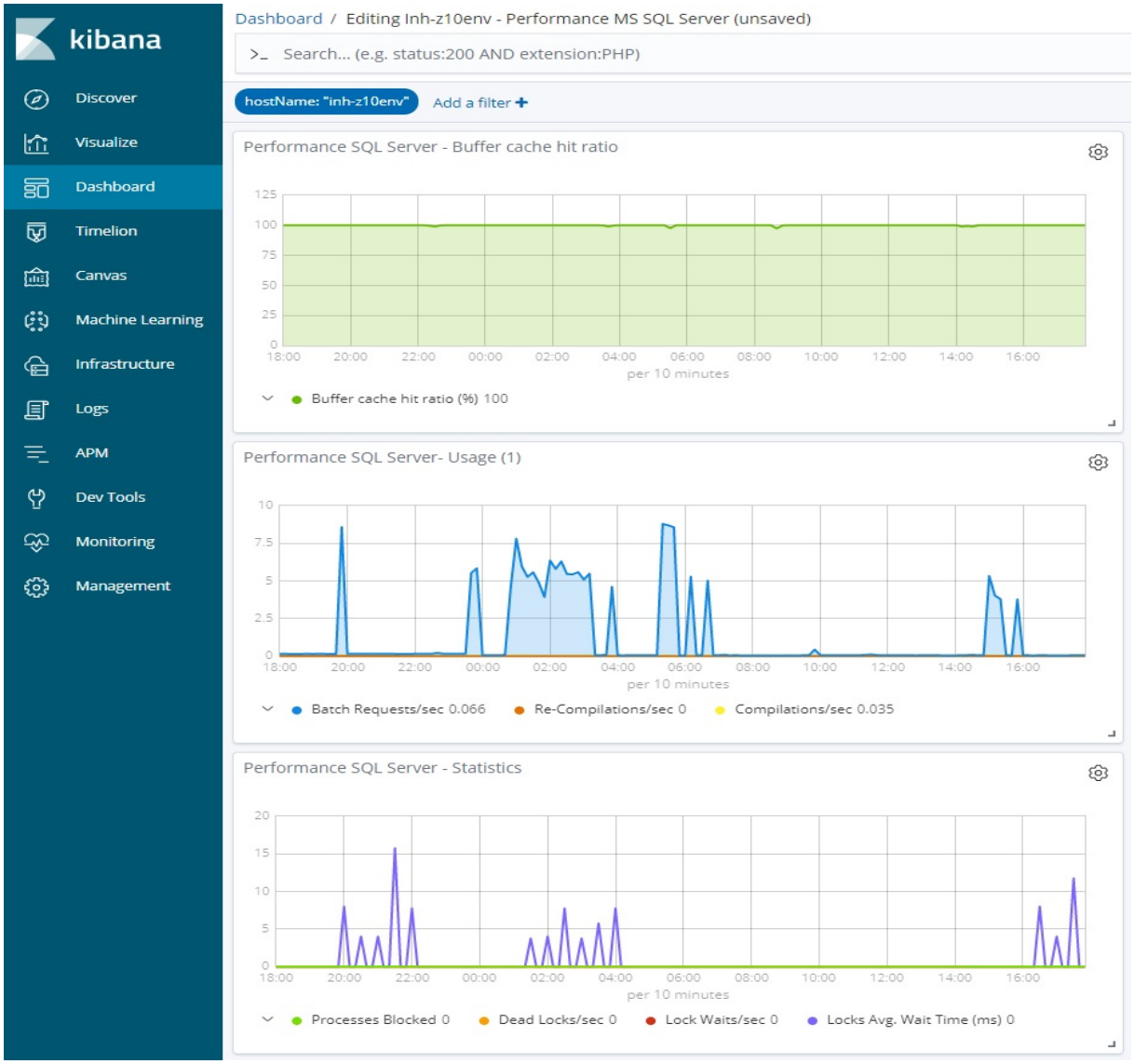
Obrázek 9: Informace o spuštěných procesech OS Windows

Detailní tabulka na obr. 9 obsahuje námi vybrané informace o procesech, jako jsou název, využití procesoru, využití operační paměti, počet vstupně-výstupních operací, počet používaných vláken, popisovačů a dobu, po jakou je proces spuštěn. Díky těmto informacím jsme byli schopni optimalizovat běh OS na počítačích a mít větší kontrolu nad spuštěnými procesy.

5.7.4 Metriky databází

Obrázek 10 obsahuje vybrané metriky, které demonstrují vytížení MS SQL Serveru v reálném čase v průběhu jednoho dne. Na základě těchto dat jsme detekovali:

- dostupnost databázového systému,
- počet dotazů, které jsou použité pro získávání dat,
- časovou prodlevu, která vzniká při nadměrném vyčítání dat.



Obrázek 10: Informace o zatížení MS SQL Serveru v průběhu jednoho dne

Námi vybrané metriky pro monitorování databází nám zajistily větší kontrolu a stabilitu v rámci celého databázové systému MS SQL Server.

6 Závěr

Během absolvování bakalářské praxe ve firmě ArcelorMittal Ostrava jsem uplatnil znalosti, které byly získany v průběhu studia. Hlavními předměty, ze kterých jsem čerpal znalosti, byly Architektura technologie .NET a Vývoj informačních systémů. Oba tyto předměty jsem absolvoval v 5. semestru studia. Informace nabyté v průběhu studia bylo možné ihned aplikovat v praxi. Díky předmětu Vývoj informačních systému jsem byl schopen návrhu architektury aplikace. Zvolená architektura byla klíčová nejen pro samotné jádro a koncept aplikace, ale také pro budoucí rozšiřitelnost, stabilitu a udržovatelnost. V implementační části jsem na základě informací z Architektury technologie .NET byl schopen implementace Windows služeb, které byly základním nástrojem v oblasti monitorování. Dále jsem využil práci s vlákny, událostmi, třídami pro diagnostiku a pokročilejší syntaxi jazyka např. linq výrazy. Zbylé informace pak poskytly předměty Algoritmy I a Algoritmy II, Programovací jazyky II, Úvod do databázových systémů a Databázové a informační systémy.

V průběhu vykonávání bakalářské praxe jsem se naučil práci s logovacími frameworky Nlog, Serilog a log4net. U log4net technologie byly získané informace nejhloubější, jelikož kromě základního použití jsem také implementoval „Appendery“ pro klienty Elasticsearch a RabbitMQ. Také jsem se naučil používat diagnostické třídy, které mi umožnily monitorovat nejen lokální počítače, ale také vzdálené servery. Dále jsem se naučil práci s technologií RabbitMQ. Technologie implementující fronty zpráv, která zajistí větší flexibilitu a nezávislost v OOP. Na závěr jsem si rozšířil znalosti v oblasti ukládání dat. Koncept ukládání dat do NoSQL databáze byl pro mě klíčovým přínosem. Při ukládání logů jsem nemusel řešit návrh a strukturu dat na straně databáze. Při vizualizování dat jsem se naučil používání nástroje Kibana, který vizualizuje získané informace, v našem případě o stavu a výkonnosti systému.

Vykonání bakalářské práce formou individuální bakalářské praxe hodnotím velmi pozitivně. V průběhu vykonávání bakalářské praxe jsem získal znalosti v oblastech logování, monitorování systému a databází. Měl jsem možnost pracovat na projektu, který má reálné využití ve firmě ArcelorMittal Ostrava. Jelikož jsem byl hlavním autorem při vypracovávání tohoto systému, tak mi byla umožněna seberealizace. Tato možnost mi přinesla značný posun v oblasti programování a analýzy systému. Rád bych uvedl, že v týmu, kde jsem pracoval, se výborně spolupracovalo a vládla kolegiální atmosféra. V případech, kdy jsem měl problémy se správným postupem, tak mi kolegové vždy ochotně pomohli a poradili. Absolvování odborné praxe ve firmě pro mě byla skvělá zkušenost a jen jsem si potvrdil, že bych v tomto oboru chtěl zůstat nadále a v budoucnu se dále rozvíjet.

Literatura

[1] C Sharp – Wikipedie. [online] [cit. 2019-01-28] Dostupné z: https://www.rabbitmq.com/

[2] Log4net Tutorial for .NET Logging: 14 Best Practices and Examples. [online] 2018 [cit. 2019-02-02] Dostupné z: https://stackify.com/log4net-guide-dotnet-logging/

[3] Messaging that just works — RabbitMQ. [online] 2007 [cit. 2019-03-11] Dostupné z: https://www.rabbitmq.com/

[4] Open Source Search & Analytics Elasticsearch | Elastic. [online] 2019 [cit. 2019-01-24] Dostupné z: https://www.elastic.co/guide/en/elasticsearch/reference/current/getting-started.html

[5] O společnosti. ArcelorMittal Ostrava a.s. [online] [cit. 2019-01-18] Dostupné z: https://ostrava.arcelormittal.com/o-spolecnosti/o-spolecnosti.aspx

[6] Windows Performance Counters Explained | AppAdminTools. [online] 2016 [cit. 2019-01-15] Dostupné z: http://www.appadmintools.com/documents/windows-performance-counters-explained/

[7] Real Time Event Log Reader - CodeProject. [online] [cit. 2019-03-21] Dostupné z: https://www.codeproject.com/Articles/1242641/Real-Time-Event-Log-Reader

[8] Programování Windows Services. [online] 2019 [cit. 2019-01-05] Dostupné z: https://www.dotnetportal.cz/clanek/194/Programovani-Windows-Services

[9] Performance Counters and their values for Performance Analysis. [online] 2019 [cit. 2019-01-10] Dostupné z: https://hexaware.com/blogs/performance-counters-and-their-values-for-performance-analysis-3/

A Implementace výkonnostních čítačů

A.1 Inicializace čítačů procesoru

```
IntanceNamesOfProcessor = new List<string>(GetInstanceNames().OrderBy(x => x));
if (Credentials != null)
{
    using (new Impersonator(Credentials.UserName, Credentials.Workstation,
        Credentials.Password))
    {
        try
        {
            TotalUtilization = new PerformanceCounter("Processor", "% Processor
                Time", "_Total", Credentials.Workstation);
            TotalPrivilegedTimeUtilization = new PerformanceCounter("Processor",
                 "% Privileged Time", "_Total", Credentials.Workstation);
            TotalUserTimeUtilization = new PerformanceCounter("Processor", "%
                User Time", "_Total", Credentials.Workstation);
            foreach (var instance in IntanceNamesOfProcessor)
            {
                UtilizationProcessors.Add(new PerformanceCounter("Processor", "%
                     Processor Time", instance, Credentials.Workstation));
            }
        }
        catch (Exception ex)
        {
            throw new Exception("Error during initialize CPU performance
                counters", new Exception(ex.InnerException.Message));
        }
    }
}
```

Výpis 3: Inicializace čítačů pro získání hodnot o výkonu procesoru ze vzdáleného PC

A.2 Získání hodnot o výkonu procesoru

```
Processor newValue = null;
try
{
    dynamic utilizationPerProcessor = new ExpandoObject();
    var dicOfUtilizationPerProcessor = (IDictionary<string, object>)
        utilizationPerProcessor;
    int instanceIndex = 0;
    foreach (var instance in IntanceNamesOfProcessor)
    {
        instanceIndex = IntanceNamesOfProcessor.IndexOf(instance);
        dicOfUtilizationPerProcessor.Add("utilizationProcessor" + instance,
        UtilizationProcessors[instanceIndex].NextValue());
    }
    newValue = new Processor()
    {
        TotalUtilization = this.TotalUtilization.NextValue(),
        TotalPrivilegedTimeUtilization = this.TotalPrivilegedTimeUtilization.
            NextValue(),
        TotalUserTimeUtilization = this.TotalUserTimeUtilization.NextValue(),
        UtilizationProcessors = utilizationPerProcessor
    };
}
catch (Exception ex)
{
    Trace.WriteLine("Error during genering new value from CPU." + " Message:" +
         Environment.NewLine + ex.Message + Environment.NewLine + "
        InnerException:" + Environment.NewLine + ex.InnerException + Environment
        .NewLine + "StackTarce:" + ex.StackTrace);
    throw new Exception(ex.Message);
}
```

Výpis 4: Získání hodnot o výkonu procesoru ze vzdáleného PC

B Implementace Windows událostních logů

B.1 Vyčítání logů

```
using (var logReader = new Log.Eventing.Reader.EventLogReader(eventsQuery))
{
    _lastReadTime = DateTime.UtcNow;

    for (EventRecord eRecord = logReader.ReadEvent(); eRecord != null; eRecord
        = logReader.ReadEvent())
    {
        containsSuccessAudit = false;

        if (eRecord.KeywordsDisplayNames != null)
        {
            containsSuccessAudit = eRecord.KeywordsDisplayNames.Contains("
                AuditSuccess") || eRecord.KeywordsDisplayNames.Contains("Úspěšný
                 audit");
        }

        if (!containsSuccessAudit)
        {
            EventLog evLogDetail = new EventLog(eRecord);
            evLogDetail.HostName = Credentials.Workstation;
            evLogDetail.HostIpAddress = GetHostIpAddress(Credentials.Workstation
                );

            OnEntryWritten(evLogDetail);
        }
    }
}
```

Výpis 5: Vyčítání Windows událostních logů

B.2 Získání informací o logu

```
private dynamic GetBasicMessage(EventRecord eventRecord)
{
    dynamic msg = new ExpandoObject();
    string fullDescription = eventRecord.FormatDescription();

    if (!string.IsNullOrEmpty(fullDescription))
        msg.Message = fullDescription;
    else
        msg.Message = string.Empty;

    return msg;
}
```

Výpis 6: Metoda pro vyčtení základních informací o logu

```
private dynamic GetFormatedMessage(XmlNode dataOfEventData, EventRecord
    eventRecord)
{
    dynamic msg = new ExpandoObject();
    var dicOfDetailedMessage = (IDictionary<string, object>)msg;

    foreach (XmlElement element in dataOfEventData)
    {
        dicOfDetailedMessage.Add(element.Attributes[0].Value, element.InnerText
            );
    }

    string descOfMessage = GetUntilOrEmpty(eventRecord.FormatDescription(), "."
        );
    msg.Message = descOfMessage;

    return msg;
}
```

Výpis 7: Metoda pro vyčtení detailních informací o logu

C Implementace ukládání logů

```
private void CreateIndexIfItDoesNotExist(string indexName)
{
    if (IndexDoesNotExist(indexName))
    {
        var createIndexResponse = Client.CreateIndex(indexName, c => c
            .Settings(s => s
            .NumberOfShards(1)
            .NumberOfReplicas(0)));

        ProcessCreateIndexResponse(createIndexResponse);
    }
}
```

Výpis 8: Metoda pro vytvoření indexu v Elasticsearch databázi

```
public void SendDocument(T obj)
{
    try
    {
        var datetime = DateTime.UtcNow.ToLocalTime();
        CreateIndexIfItDoesNotExist(IndexName);

        var response = Client.Index(obj, i => i.Index(IndexName));
        ProcessIndexResponse(response);
    }
    catch (Exception ex)
    {
        Trace.WriteLine("Error in ES client during sending document" + "
            Message:" + Environment.NewLine + ex.Message + Environment.NewLine +
             "InnerException:"
        + Environment.NewLine + ex.InnerException + Environment.NewLine + "
            StackTarce:" + ex.StackTrace);
    }
}
```

Výpis 9: Metoda pro uložení výsledného logu do Elasticsearch databáze